The Gravity Engine: A Mathematical Framework for AI Decision Routing

Abstract

Autonomous systems make an unending stream of small routing decisions: which path to take, whether an action is permitted, how much to trust a pattern that worked before, how to recover from failure. Ad hoc, these decisions accrete into unmaintainable heuristics. We describe the **Gravity Engine**, a deliberate attempt to give an AI-operations platform a *single mathematical vocabulary* for these decisions, composed of six well-understood formalisms borrowed from optimisation, biology, and network theory: **Bellman optimality** (decision routing), **Physarum dynamics** (reinforcement/decay of paths), a **trust gradient** (autonomy weighting), **entropy minimisation** (self-healing), **network value** (knowledge-capture prioritisation), and **hexagonal coverage** (completeness). The contribution of this paper is not the formulas — each is standard — but (1) their *composition* into one control layer with a clear division of labour, and (2) a rigorous, self-critical accounting of *which formalisms are load-bearing in production versus which are organising metaphors*. We find the framework's honest value is bimodal: two of the six (Bellman-style cost routing with an infinite-cost policy barrier, and a Physarum-style reinforcement of proven paths) map onto real, consequential control decisions and are worth the formalism; the other four function as valuable *design heuristics* — they shape how the system is built and reasoned about — but are not, in their literal mathematical form, the mechanism that runs in production. We argue this distinction is the point: a framework's usefulness is not diminished by admitting which parts are engine and which are compass.

1. Motivation: the vocabulary problem

A large autonomous platform (this one: 805 services, 2,688 endpoints) makes routing decisions everywhere — model selection, action permission, failure recovery, escalation. Written ad hoc, each decision is a local heuristic; collectively they are unreviewable. The Gravity Engine is a bet that a *shared mathematical vocabulary* for these decisions is worth more than locally-optimal heuristics: it makes decisions comparable, auditable, and composable, even where the underlying formula is approximate.

The bet has a well-known risk — **formalism theatre**, where equations decorate decisions they do not actually govern. The main methodological commitment of this paper is to *audit our own framework for exactly this*, and to report the result honestly rather than defend all six formulas equally. A framework you cannot criticise is a brand, not a tool.

We present each formalism with its formula, its intended role, and a verdict: **engine** (load-bearing control), **compass** (a design heuristic that shapes the build without being the literal runtime mechanism), or **unproven**.

2. The six formalisms

2.1 Bellman optimality — decision routing · verdict: engine

$$C(\text{action}) = w_r \cdot \text{risk} + w_t \cdot \text{time} + w_e \cdot \text{effort} + \infty \cdot \mathbb{1}[\text{policy violation}]$$

$$V^*(s) = \min_a \left[C(s,a) + \gamma V^*(s') \right]$$

with weights (risk, time, effort) = (0.4, 0.3, 0.3). The intended role is to select the lowest-cost *valid* path for any dispatch, with two features that make it more than decoration: **reversible actions halve the risk weight and irreversible actions double it** (so the cost function encodes a real asymmetry the system must respect), and **a policy violation contributes infinite cost** — an absolute barrier the optimiser can never traverse.

Why engine: the infinite-cost policy barrier is not a metaphor; it is the mathematical form of a hard governance rule (some actions are never permitted regardless of how cheap they look), and it composes cleanly with the finite cost terms — a single `min` over `risk/time/effort + ∞·violation` correctly refuses forbidden actions while ranking permitted ones. The risk-weight asymmetry for (ir)reversibility is a real, consequential parameter. The Bellman *recursion* over future value (`γ·V*(s')`) is the more aspirational part — full look-ahead is rarely computed — but the *one-step* cost minimisation with the ∞ barrier is genuinely how permissioned routing should be expressed, and expressing it this way makes every routing decision auditable as "the lowest-cost action not behind an infinite barrier."

2.2 Physarum dynamics — path reinforcement · verdict: engine (in spirit)

$$\frac{dP}{dt} = \alpha |F| - \beta P \qquad (\alpha = 0.3, \beta = 0.05)$$

Borrowed from the slime-mould *Physarum polycephalum*, which reinforces tubes carrying more flux and lets unused tubes atrophy. Here P is a path's conductance, F its recent flux (successful use): successful patterns gain conductance (\alpha|F|) and unused ones decay (\beta P). The intended role is to make the system *prefer paths that have worked* and *re-validate paths that have gone stale*.

Why engine (in spirit): the platform genuinely has a flow-control layer that admits, delays, or absorbs work based on a conductance-like signal, and the *principle* — reinforce what works, decay what is unused, re-validate the stale — is a real, load-bearing behaviour (proven patterns get more autonomy; unused code paths are treated as suspect). The literal ODE with those exact constants is an idealisation of the discrete update the system actually performs, so we grade it "engine in spirit": the reinforcement/decay *dynamics* are real control; the specific \alpha, \beta are chosen, not fitted.

2.3 Trust gradient — autonomy weighting · verdict: compass

$$\text{Trust} = 0.7 \cdot \text{recent evidence} + 0.3 \cdot \text{historical pattern}$$

Recency-weighted trust that governs how much autonomy a pattern earns: proven-recently → auto-proceed on reversible actions; older or new → require approval. The intended role is a smooth dial from "verify everything" to "act freely."

Why compass: the *principle* is real and shapes the system (recent evidence dominates; new patterns require verification), but the specific 0.7/0.3 convex combination is an assumed weighting, not a measured one, and — candidly — a forensic finding on this very platform showed a trust score **hard-coded at a constant** in a live path rather than computed from this formula. That is exactly the formalism-theatre risk §1 warned of: the equation existed in doctrine while a constant ran in production. We grade it compass: valuable as a design principle for *how* autonomy should scale with evidence, not (yet) a faithfully-implemented runtime computation.

2.4 Entropy minimisation — self-healing · verdict: compass

$$H = \lambda \cdot (\text{target} - \text{current})$$

A control-theoretic restoring force: on failure, route toward alternatives proportional to the gap between the target and current state, so the system self-organises back toward order rather than retrying the failed path. The intended role is failure recovery that *diversifies* rather than repeats.

Why compass: the behaviour it prescribes — on failure, do not retry the same thing; route to an alternative proportional to the disorder — is a genuinely good and observable design discipline (the platform's recovery logic does prefer alternatives over retries). But H = \lambda(\text{target}-\text{current}) is a proportional-control *schema*, not a specific implemented controller with a measured \lambda; it organises the recovery design without being the literal runtime equation. Compass.

2.5 Network value — knowledge-capture priority · verdict: compass

$$V(n) = k \cdot n \cdot \log(n)$$

A super-linear valuation of accumulated knowledge (each fact captured makes existing facts more valuable), used to prioritise *capturing* knowledge over other work. The intended role is to justify aggressive knowledge/evidence capture.

Why compass: the platform does capture prodigiously — a **~40-million-row evidence substrate, 252 artifact types, ~64,000 artifacts/day** — and the *super-linear* intuition (a knowledge graph's value grows faster than its node count) is a reasonable organising principle. But nothing in production computes $k \cdot n \log n$ to make a decision; the formula is a *rationale* for a capture-everything policy, not a runtime valuation. Compass — and a useful one, because it correctly predicts that the binding problem becomes *querying* the captured knowledge efficiently (the subject of the companion wieldability paper), not capturing it.

2.6 Hexagonal coverage — completeness · verdict: compass

A feature is "complete" only if six neighbours are covered: **tests, security, UX, docs, performance, observability**; and no two concepts should be more than six hops apart. The intended role is a completeness checklist that resists half-built features.

Why compass: this is a discipline, not a computation — a hexagon as a mnemonic for "did you cover all six facets." It has clearly shaped the platform (the six facets recur in how work is scoped), and it is valuable *as* a checklist. Calling it "mathematics" is generous; calling it a rigorous, load-bearing design heuristic is fair.

---

3. The honest ledger

Formalism	Role	Verdict
Bellman cost + ∞ policy barrier	decision routing	**engine**
Physarum reinforce/decay	path preference	**engine (in spirit)**
Trust gradient (0.7/0.3)	autonomy weighting	compass (found hard-coded in one live path)
Entropy $\lambda(\text{target}-\text{current})$	self-healing	compass
Network value $k\,n\log n$	capture priority	compass
Hexagonal 6-neighbour	completeness	compass (design checklist)

Two engines, four compasses. We consider this a *successful* audit, not a disappointing one. The two engine formalisms — permissioned cost minimisation with an infinite barrier, and reinforce-what-works/decay-what-is-stale — are exactly the two decisions a routing-and-governance layer most needs to make *rigorously*, and expressing them mathematically buys real auditability. The four compasses are genuinely useful design heuristics that shaped a large, coherent system; presenting them as literal runtime mathematics would be the formalism theatre we set out to avoid.

---

4. Discussion

The value of a unified vocabulary is compositional. Because permission, cost, trust, and recovery are all expressed in the same idiom, a single dispatch decision can be read as: *minimise (risk, time, effort) over actions not behind an infinite policy barrier, biased toward paths with high conductance, with autonomy scaled by recency-weighted trust, and on failure route by the entropy gradient toward an alternative.* Whether every clause of that sentence is a faithful computation (two are) or a design commitment (four are), the sentence is *auditable* — a reviewer can ask, of any decision, which term dominated and why. That auditability is the real product of the framework, and it survives the honest downgrading of four of the six formulas.

The broader methodological point generalises beyond this system: **a control framework earns credibility precisely by distinguishing its engines from its compasses.** Frameworks that insist all their equations are load-bearing invite the reasonable suspicion that none are. We have tried to demonstrate the opposite discipline.

Limitations

This is an audit of one system by an author inside it; the engine/compass verdicts are argued from how the code behaves, not from an independent instrumentation of which formula fired on which decision (which would be the rigorous next step and is not done here). The weights and constants (0.4/0.3/0.3; 0.7/0.3; α=0.3, β=0.05) are chosen, not fitted to data — a genuine weakness for anything claiming to be "the mathematics," and the honest reason four formalisms are graded compass. The trust-gradient finding (a hard-coded constant where the formula was doctrine) is a single observed instance, not a systematic audit of every formula's fidelity to its

implementation — there may be more formalism theatre we did not catch. Finally, "engine vs compass" is our own binary imposed for clarity; some formalisms (Physarum) sit genuinely between, and we have flagged where.

---

5. Conclusion

The Gravity Engine set out to give an autonomous platform one mathematical vocabulary for its endless small decisions. Audited honestly, two of its six formalisms are load-bearing control — Bellman cost minimisation with an infinite policy barrier, and Physarum-style reinforcement of proven paths — and four are valuable design compasses that shaped the system without being its literal runtime mathematics. The result is not a weaker framework but a more credible one: its usefulness lies in the auditability a shared vocabulary buys, and its integrity lies in saying which parts are engine and which are compass. A framework worth trusting is one that can tell you where it is only a metaphor.

Appendix A — Constants and grounding

Item	Value / source
Bellman weights (risk, time, effort)	0.4, 0.3, 0.3 (chosen)
Reversibility risk-weight rule	halve if reversible, double if irreversible
Physarum (α reinforce, β decay)	0.3, 0.05 (chosen)
Trust gradient (recent, historical)	0.7, 0.3 (chosen; found hard-coded in one live path)
Evidence substrate (network-value grounding)	~40 M rows, 252 types, ~64,000/day
Platform scope the framework governs	805 services, 2,688 endpoints
Formalisms graded engine / compass	2 / 4

Constants are design parameters, not fitted values. Grounding figures measured live, 23 July 2026.