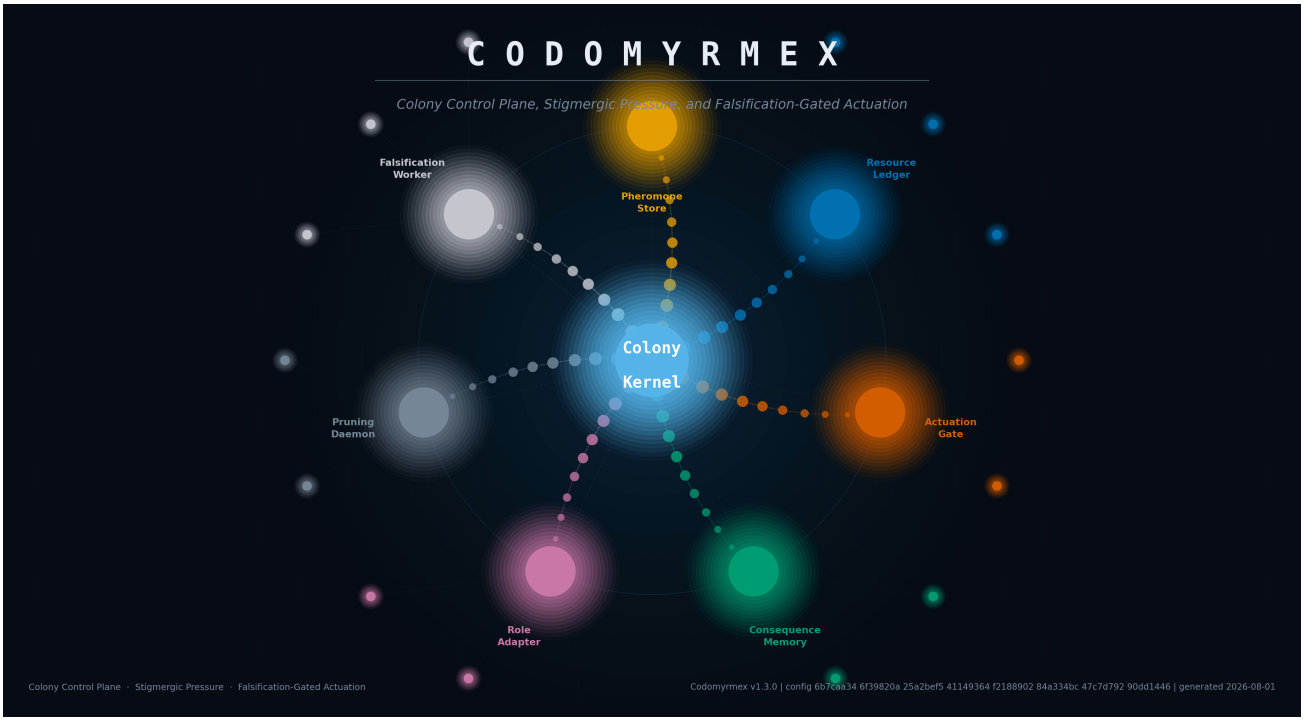

Codomyrmex: An Artificial Ecology for Agentic Software Development

Colony Control Plane, Stigmergic Pressure, and Falsification-Gated Actuation

Daniel Ari Friedman

Date: August 1, 2026

ORCID: 0000-0001-6232-9096

DOI: 10.5281/zenodo.21750800

Repository: github.com/docxology/codomyrmex

Contents

Abstract

Agentic software can preserve task state while still forgetting the consequences of prior actions. Codomyrmex studies a narrow control-plane question: after a caller reports a failed action at one software location, can the system deterministically increase friction for a materially similar proposal at that location without changing an unrelated target? Its Colony Control Plane records consequence reports and couples them to target-indexed signal pressure, agent trust, role labels, resource accounting, adversarial checks, and an explicit EXECUTE/HOLD/REFUSE gate.

The implementation comprises 8 cooperating subsystems. The ordinary Model Context Protocol path remains caller-reported and unattested. Optional and required `ColonyKernel` attestation modes instead bind proposal, verdict, authorization, execution receipt, and outcome in a signed, hash-linked local ledger. That ledger protects lifecycle linkage but does not independently observe external actuation or establish deployment safety. Consequence records can use file-backed SQLite; the default MCP kernel and signal field remain process-local.

Evaluation is limited to implementation properties and controlled fixtures. At composition time, the scoped Colony Kernel surface contains 819 passing tests with 76.6% branch coverage, 0 Ruff errors, and 0 ty diagnostics. A paired deterministic replay moves the same-target proposal from 0.875/EXECUTE to 0.725/HOLD after a reported failure while leaving an unrelated target unchanged. Separate fixtures exercise trust promotion, bounded arithmetic, linear signal decay, local attestation integrity, and interface behavior. These results support reproducible software contracts, not ecological optimality, calibrated risk, production harm reduction, or generalization to external workloads.

The report contributes the typed control plane, transparent gate, coupled local feedback, authenticated local lifecycle option, and source-bound publication workflow. Generated variables, figures, citations, claim boundaries, and release receipts tie the rendered report to the evaluated checkout. End-to-end external-actuation attestation, restart-persistent field storage, representative benchmarks, and independent deployment validation remain open.

Keywords: ai-agents, model-context-protocol, mcp, multi-agent, orchestration, colony-control-plane, stigmergy, artificial-ecology, agentic-software-engineering, falsification-worker, actuation-gate, trust-scoring

Corresponding author: Daniel Ari Friedman

1 Introduction: A Bounded Control-Plane Question

Tool-using language-model agents can browse, retrieve, call APIs, edit repositories, and run software (Nakano et al. 2021; Karpas et al. 2022; Yao et al. 2023; Schick et al. 2023; Qin et al. 2023; Patil et al. 2023). Once an agent can alter persistent state, success is not only a planning problem. It is also an authorization, accounting, and feedback problem: what evidence should be inspected before an action, what should be recorded afterward, and how should that record affect the next proposal?

Long-horizon environments and software-agent benchmarks increasingly evaluate stateful, multi-step interaction rather than isolated text generation (Yang et al. 2024b, 2024a; Liu et al. 2023; Zhou et al. 2023; Xie et al. 2024; Trivedi et al. 2024). Security evaluations likewise treat tool misuse, prompt injection, privacy leakage, and trajectory-level effects as system properties (Greshake et al. 2023; Debenedetti et al. 2024; Ruan et al. 2023; Zhang et al. 2024). These results motivate explicit controls between a model's proposal and consequential actuation.

1.1 Problem statement

Contemporary orchestration frameworks provide routing, checkpoints, memory, roles, and multi-agent coordination (AI 2024; Wu et al. 2023; Inc. 2024). Codomyrmex does not claim those systems lack state. It targets a narrower integration question:

> Can a recorded failure at one software location deterministically increase the admission cost of a later proposal at that same location, without changing the later model's context or weights?

The question is deliberately local and falsifiable. A convincing positive result requires a paired case: hold agent, proposal, and budget factors fixed; add the failed outcome at one target; show that the same-target score does not increase and that an unrelated target is unchanged.

1.2 Bounded ecology thesis

Codomyrmex uses "colony" and "pheromone" as engineering metaphors for a shared control plane. Stigmergy describes coordination mediated through changes to a shared environment rather than direct pairwise messages (Grassé 1959; Parunak 1997; Bonabeau et al. 1999). In the implementation, the shared environment is a typed, process-local signal field. `FAILURE`

records caller-reported adverse outcomes, RISK records prospective concerns, and the gate scores their maximum at the proposal target.

The project specification's design criterion is that the colony should become "harder to fool after every failed action." This manuscript narrows that phrase to an implemented contract:

- a canonical failed report deposits same-target FAILURE pressure;
- effective local hazard is `max(RISK, FAILURE)`;
- higher hazard cannot increase the ordinary score;
- the paired lower-trust case moves from 0.875/EXECUTE to 0.725/HOLD;
- an unrelated target remains unchanged; and
- passive decay eventually removes the added friction.

This does not make the system deception-proof. The ordinary MCP path accepts caller-reported outcomes without attesting them against a prior EXECUTE authorization. Optional and required `ColonyKernel` attestation modes locally bind the lifecycle from proposal through outcome, but the ledger does not independently observe external actuation or establish deployment safety. The default field and consequence database are also in-memory and disappear on process restart. The paired feedback claim is therefore process-local, report-dependent, and reversible.

1.3 Contribution

The paper contributes five concrete artifacts.

1. A Colony Control Plane. 8 named subsystems separate typed signal storage, resource accounting, actuation gating, consequence records, role inference, pruning nomination, deterministic falsification, and integration.
2. A transparent ternary gate. Budget, effective local hazard, trust credit, and proposal completeness form a bounded weighted score, subject to explicit hard overrides and EXECUTE/HOLD/REFUSE routing.
3. Coupled local feedback. Reported FAILURE and prospective RISK remain separately inspectable but jointly constrain the gate through their maximum; the integrated gate also reads recent failures from the kernel's consequence memory.
4. A real contract suite. Tests use real subsystem instances to establish same-target inhibition, cross-target isolation, linear decay recovery, score bounds, trust updates, and interface behavior.
5. A source-bound report and release route. Repository-root commands regenerate variables and figures, render semantic HTML plus content and distribution PDFs, validate claims and citations, and prepare a detached-hash publication bundle.

The contribution is a reference implementation and evidence boundary, not a completed production-security system.

The numeric values shown here are current implementation defaults or illustrative initial settings, not universal constants, fitted parameters, or empirical estimates. They are configurable and can be tuned or replaced through the corresponding runtime policy, configuration, or presentation inputs; after a change, regenerate the tests, tables, figures, and manuscript before interpreting the result. This distinction lets the implementation be reproducible without turning a reproducible fixture into a scientific calibration claim.

1.4 Architecture in brief

`ColonyKernel.propose_action` first runs deterministic falsification, checks the resource budget, loads the agent profile, refreshes its role label, and asks `ActuationGate` for a decision. Ordinary scoring uses

- binary budget approval (weight 0.3);
- tiered credit from effective hazard `max(RISK, FAILURE)` (weight 0.3);
- tiered trust credit, optionally reduced after 3 recent failures (weight 0.25); and
- completeness of rollback, evidence, and expected outcome fields (weight 0.15).

Budget failure, SANDBOX, trust below 0.3, and CRITICAL falsification are early returns. The higher role labels are inferred trust tiers and intended specializations; the current gate does not enforce a complete action-by-role permission matrix.

`record_outcome` is a separate caller operation. It updates the consequence store, resource ledger, trust profile, role label, and signal field. A failed test report deposits a FAST FAILURE trace; a clean report reinforces/deposits SUCCESS. On the ordinary MCP path, proposal and outcome are not linked by a consumed authorization ledger, so "outcome" means a submitted report unless explicitly qualified. In optional or required attestation mode, the kernel locally requires and consumes the linked authorization/receipt chain; this authenticates ledger linkage, not external execution.

The 8 MCP tools expose this stateful kernel through JSON-shaped requests and responses. A module-level singleton shares state across calls in one server process. File-backed SQLite can persist consequence records when configured, but the field has no restart-persistent backend.

1.5 Relation to prior work

The implementation combines ideas from several established areas without claiming to replace them:

- multi-agent systems treat autonomy, reactivity, proactivity, and social interaction as engineered properties (Wooldridge and Jennings 1995);
- computational trust and reputation update beliefs or scores from interaction history (Marsh 1994; Sabater and Sierra 2005; Kamvar et al. 2003);
- runtime-assurance and shielding research inserts a safety decision layer between an advanced controller and actuation (Seto et al. 1998; Alshiekh et al. 2018);
- least privilege, capability security, and zero-trust architecture motivate explicit, repeatedly evaluated authority boundaries (Saltzer and Schroeder 1975; Miller and Shapiro 2003; Rose et al. 2020); and
- reproducible research, model reporting, and assurance cases motivate traceable claims and explicit limitations (Peng 2011; Mitchell et al. 2019; Raji et al. 2020; Buhl et al. 2024).

Codomyrmex's specific combination is a target-indexed signal field coupled to a transparent software-actuation gate and consequence ledger. Comparative superiority over other frameworks is not established in this release.

1.6 Evidence boundary

The executed evidence is the scoped Colony Kernel quality gate and deterministic fixtures. The proposed 4-condition, 20-run benchmark has not been executed. No population refusal rate, throughput advantage, production harm reduction, or long-run convergence claim is reported. The paper treats ordinary-path unlinked outcome reporting, external-actuation attestation, SANDBOX bootstrap, persistence, role permissions, and external calibration as open engineering work.

1.7 Reader's guide

Section 2 states only the invariants supported by the runtime recurrence and gate arithmetic. Section 3 describes subsystem behavior and call sequencing. Section 4 separates the proposed external study from the live configuration. Section 5 reports executed gates, deterministic fixtures, and analytical score cases. Section 6 compares related systems and states limitations. Section 7 offers a bounded conceptual analogy rather than a formal equivalence. Section 8 documents the build and evidence chain. Section 9 summarizes the supported claim and next tests. Section 13 records design alternatives and remaining tradeoffs.

2 Operational Semantics and Verified Invariants

This section states the mathematical properties that follow from the checked-in implementation. It deliberately separates three kinds of statement:

1. implementation invariants, which follow directly from the recurrence and gate arithmetic;
2. deterministic contract cases, which are exercised by tests; and
3. hypotheses, which require external workloads or calibrated outcome data.

The gate score is a design score, not a probability of safety. The trust process is a bounded accounting rule, not a Bayesian posterior. No claim below establishes production safety, optimality, differential privacy, or long-run ecological convergence.

The numeric values shown here are current implementation defaults or illustrative initial settings, not universal constants, fitted parameters, or empirical estimates. They are configurable and can be tuned or replaced through the corresponding runtime policy, configuration, or presentation inputs; after a change, regenerate the tests, tables, figures, and manuscript before interpreting the result. The equations in this section state consequences of the current implementation; they are not a claim that the selected constants are universal or empirically calibrated. Notation is fixed in Section 12; in particular, h denotes hazard, ρ denotes risk clearance, and f_n denotes human feedback.

2.1 Capped local signal field

Let $J_t \subseteq \mathcal{L} \times \mathcal{K}$ be the finite set of compound location–signal keys present at tick t, where $\mathcal{K}$ contains `FAILURE`, `SUCCESS`, `RISK`, `NEED`, `DEPENDENCY`, and `HUMAN_PRIORITY`. Over any finite analysis horizon, choose a finite universe J containing the keys that appear and represent an absent or deleted key by zero. The projected field state is then

$$x_t \in [0.0, M]^J, \qquad M = 10.0. \tag{1}$$

This capped non-negative cube is a complete metric space under the supremum metric and a complete lattice under the coordinate-wise order. It is not a vector space: negative scaling and unrestricted addition leave the state space.

For key j, let $\epsilon_{j,t} > 0$ be the evaporation amount stored for that key during step t, and let $d_{j,t} \geq 0$ be the effective deposits applied during the step, after source and trust multipliers. A redeposit may replace the stored decay class; when it does not, write the constant amount as ϵ_j. With the convention "evaporate, then deposit," the implemented update can be represented as

$$x_{j,t+1} = \min\big(M,\ \max(0.0, x_{j,t} - \epsilon_{j,t}) + d_{j,t}\big)\,. \tag{2}$$

The 3 configured evaporation amounts are $\epsilon_{\text{FAST}} = 0.3$, $\epsilon_{\text{NORMAL}} = 0.1$, and $\epsilon_{\text{SLOW}} = 0.02$ strength units per tick. A caller may deposit at another point in the scheduler cycle, so exact within-tick values depend on operation order; the range and monotonicity results below do not.

Lemma 1 (range invariance). If $x_{j,0} \in [0.0, M]$, then $x_{j,t} \in [0.0, M]$ for every finite sequence of non-negative deposits.

Proof. The inner maximum is non-negative and the outer minimum is at most M. Induction over t completes the argument. □

Lemma 2 (passive linear decay). With no deposits and a fixed stored decay amount ϵ_j for n ticks,

$$x_{j,t+n} = \max(0.0, x_{j,t} - n\epsilon_j). \tag{3}$$

Proof. Apply the subtract-and-floor operation repeatedly. Before the first floor event, exactly ϵ_j is removed per tick; after the value reaches zero, further applications leave it at zero. □

A trace with current strength $x_{j,t}$ therefore disappears after at most

$$N_{\text{extinct}}(x_{j,t}, \epsilon_j) = \left\lceil \frac{x_{j,t}}{\epsilon_j} \right\rceil \tag{4}$$

ticks without reinforcement. This is finite forgetting, not exponential half-life. For a unit trace, FAST, NORMAL, and SLOW disappear after 4, 10, and 50 discrete ticks, respectively (the ceiling matters when $x_{j,t}/\epsilon_j$ is not integral).

All traces begin at strength 1.0 and decline linearly toward 0.0. FAST has the steepest slope and earliest vertical extinction marker, NORMAL is intermediate, and SLOW extends furthest to the right. Line style, endpoint label, and colour redundantly identify each class; the curves are analytical rather than observed time series.

2.2 From reported failure to local gate pressure

The witness retains prospective `RISK` and caller-reported `FAILURE` as separate channels. The gate uses their maximum as effective local hazard pressure:

$$h_t(\ell) = \max\Big(x_{(\ell,\text{RISK}),t}, x_{(\ell,\text{FAILURE}),t}\Big)\,. \tag{5}$$

The risk-clearance component is the piecewise function

$$\rho(h) = \begin{cases} 1.0, & 0.0 \leq h < 3.0, \\ 0.5, & 3.0 \leq h < 6.0, \\ 0.0, & h \geq 6.0. \end{cases} \tag{6}$$

Proposition 1 (local monotonicity). Holding all other gate inputs fixed, increasing either `RISK` or `FAILURE` pressure at the proposal target cannot increase the gate score.

Proof. The maximum in Equation 5 is coordinate-wise non-decreasing, while $\rho(h)$ in Equation 6 is non-increasing. Its gate coefficient is positive. □

Deterministic paired case. A caller-reported failed outcome deposits a base-strength 2.0 `FAILURE` signal from source `TEST`. The source multiplier 1.5 produces effective pressure 3.000. For an authorized lower-tier agent with a complete proposal, clear budget, and no prior pressure, the ordinary score changes from

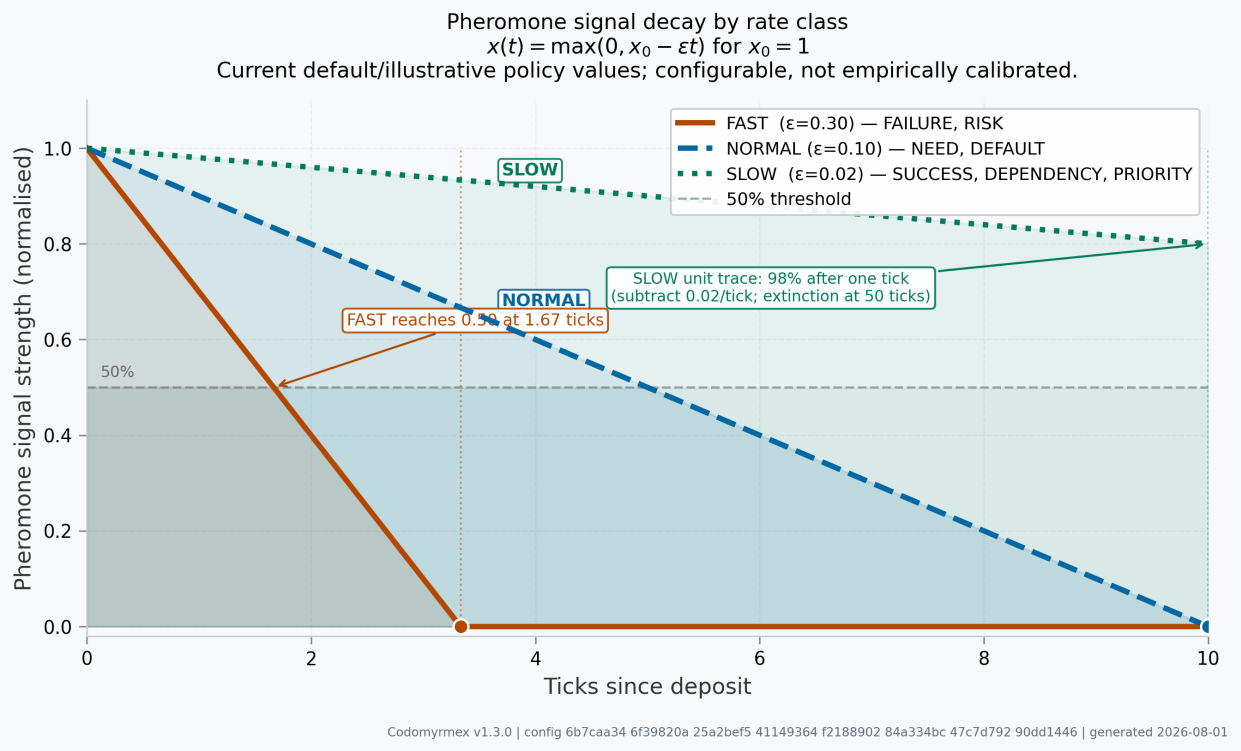

Figure 1: Analytical trajectories of a 1.0-strength trace under the implemented subtractive decay rule. Each curve is $x_{j,t} = \max(0.0, x_{j,0} - \epsilon_j t)$, with no deposits, reads, or reinforcement. Vertical markers show continuous extinction points; the runtime updates at integer ticks and removes values at or below zero. Distinct dash patterns and direct endpoint labels identify the decay classes without relying on colour. Current default/illustrative policy values; configurable, not empirically calibrated. Parameters are generated from the manuscript snapshot.

$$g_{\text{clear}} = w_b b + w_\rho \rho(h_{\text{clear}}) + w_u u + w_c c = w_b(1) + w_\rho(1) + w_u(0.5) + w_c(1) = 0.875 \tag{7}$$

to

$$g_{\text{failed}} = w_b b + w_\rho \rho(h_{\text{failed}}) + w_u u + w_c c = w_b(1) + w_\rho(0.5) + w_u(0.5) + w_c(1) = 0.725. \tag{8}$$

Thus the same-target decision changes from EXECUTE to HOLD, while an otherwise identical proposal at another target remains at 0.875. An executed integration test checks all three claims. Passive decay eventually restores the clear-field score. This is the precise implemented meaning of "harder to pass after local failure"; it is neither permanent nor a proof that deception is prevented.

2.3 Gate arithmetic and hard overrides

For proposals that reach ordinary scoring,

$$g = 0.3b + 0.3\rho(h) + 0.25u + 0.15c, \tag{9}$$

where $b \in \{0.0, 1.0\}$, $\rho(h) \in \{0.0, 0.5, 1.0\}$, u uses the generated trust-credit tiers and optional 0.25 recent-failure penalty, and c is generated from zero through 3 missing fields with a 0.35 penalty per field.

Four checks can bypass ordinary thresholding:

- an unaffordable proposal returns HOLD in the integrated kernel and REFUSE in standalone gate use;
- `SANDBOX` returns REFUSE;
- trust below 0.3 returns REFUSE; and
- a CRITICAL falsification finding returns REFUSE.

Otherwise $g \geq 0.75$ yields EXECUTE, $0.5 \leq g < 0.75$ yields HOLD, and $g < 0.5$ yields REFUSE.

Proposition 2 (score boundedness). Every ordinary score is in $[0.0, 1.0]$.

Proof. Every component is in $[0.0, 1.0]$, every coefficient is non-negative, and the coefficients sum to 1.0. The implementation additionally clips the sum to $[0.0, 1.0]$. □

Proposition 3 (component monotonicity). On the ordinary path, improving budget approval, risk clearance, trust credit, or completeness while holding the other components fixed cannot reduce g. Increasing local hazard or activating the recent-failure penalty cannot increase g.

These are arithmetic monotonicity statements. They do not imply that g is calibrated to real-world harm. Such calibration would require linked proposals, independently observed and externally attested outcomes, representative workloads, and held-out evaluation.

Figure 2 visualizes the exact score tiers and the continuous completeness envelope used only to make the discontinuities legible.

The main panel rises across completeness and changes in discrete tiers across trust, with horizontal threshold planes marking HOLD and EXECUTE. A companion projection uses distinct line styles, point shapes, direct labels, and colour for selected trust slices. The continuous completeness surface is a visual envelope over a runtime input that is actually discrete.

2.3.1 What HOLD can and cannot establish

A third decision can be useful when revision supplies information, but it does not automatically dominate a binary gate. Let $L_E(z)$ and $L_R(z)$ be expected losses from EXECUTE and REFUSE at observed state z. Let C_H be revision cost and let Z' be the evidence available after revision. HOLD has lower expected loss only when

$$C_H + \mathbb{E}[\min\{L_E(Z'), L_R(Z')\} \mid z] < \min\{L_E(z), L_R(z)\}. \tag{10}$$

The current implementation returns actionable evidence requests, but it does not estimate any term in Equation 10. HOLD is therefore an auditable design choice and a future empirical hypothesis, not a proved optimum.

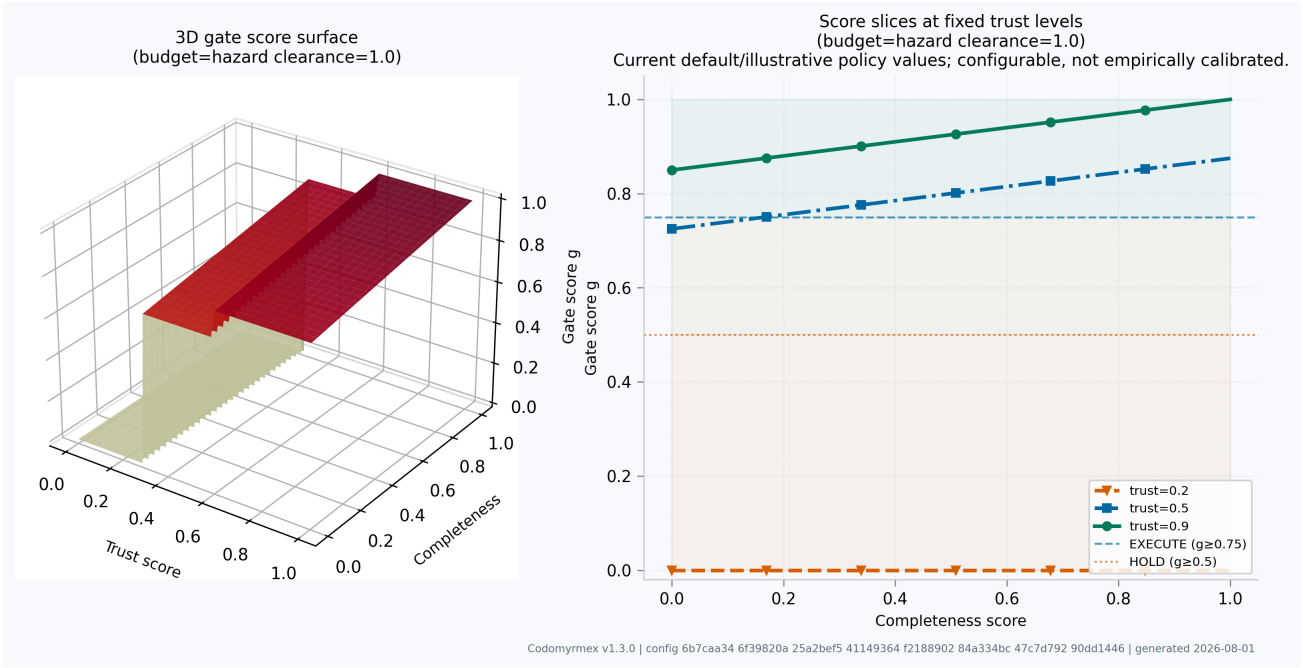

Figure 2: Analytical gate-score surface with budget and local hazard clearance fixed at 1.0. The implementation's trust hard floor and trust-credit tiers are shown; completeness is plotted continuously as a visual envelope even though runtime completeness is discrete. Threshold planes identify the configured HOLD and EXECUTE boundaries. This is formula-derived, not an empirical risk surface. Current default/illustrative policy values; configurable, not empirically calibrated.

2.4 Trust as bounded evidence accounting

For recorded outcome n, trust follows

$$\tau_{n+1} = \text{clip}_{[0.0,1.0]}(\tau_n + \Delta_n), \tag{11}$$

with

$$\delta_{\text{test}}(n) = \begin{cases} +0.04, & \text{tests pass}, \\ -0.08, & \text{tests fail} \end{cases}, \qquad \delta_{\text{repair}} = -0.05, \qquad \Delta_n = \delta_{\text{test}}(n) + \delta_{\text{repair}}\, 1_{\text{repair}}(n) + 0.03 f_n, \qquad f_n \in [-1.0, 1.0]. \tag{12}$$

This proves boundedness by construction. It does not prove convergence. With continuing random non-zero increments, a clipped constant-step process can continue moving indefinitely.

If tests pass independently with probability p_{pass}, with no repairs and neutral human feedback, the expected *unclipped* increment is

$$\mathbb{E}[\Delta] = 0.04 p_{\text{pass}} + -0.08(1 - p_{\text{pass}}). \tag{13}$$

The drift is zero at $p = 0.667$, positive above it, and negative below it. This is a balance point for expected increments—not a unique trust equilibrium—because the update contains no state-dependent restoring term.

Starting from $\tau_0 = 0.1$, an all-success path adds 0.04 per outcome until clipping. The first 3 recorded successes satisfy the generated promotion contract for `REPAIR_ANT`. This is a deterministic path claim, not an expected hitting time for an imperfect agent. Moreover, the ordinary MCP path accepts caller-reported outcomes without linking them to a prior EXECUTE decision. Optional and required local attestation modes can enforce lifecycle linkage, but they do not make a submitted outcome an independent observation of competence. The three-record bootstrap must therefore be treated as supervised evidence, not autonomous proof of competence.

The integrated gate now holds a reference to the same `ConsequenceMemory` used by the kernel. 3 recent failures reduce trust credit by 0.25 before weighting. This wiring property and its score effect are contract-tested.

2.5 Privacy and integrity boundary

The released mechanism is deterministic and does not provide differential privacy. Under replacement adjacency, one record can change a trust increment from its maximum $+0.07$ to its minimum -0.16, so the one-step global sensitivity is at most

$$\Delta_{\text{replace}} = 0.23. \tag{14}$$

Clipping may reduce the realized difference near a boundary but does not lower this global bound. Any future noisy release would also need an explicit adjacency relation, composition accounting across repeated queries, and a utility analysis (Dwork and Roth 2014).

The more immediate integrity issue is the boundary between ledger authentication and external observation. The ordinary `colony_record_outcome` path constructs a proposal from caller input and requires no outstanding, matching, previously EXECUTEd proposal. Optional or required kernel attestation instead binds and consumes a local authorization/receipt chain. Even there, the ledger authenticates submitted lifecycle events; it does not independently observe the external action or establish that the action was safe. Trust and local pressure therefore remain evidence about recorded events, not independently verified ground truth. This limitation bounds every causal and security claim in the manuscript.

2.6 Verified claims and open hypotheses

Table 1 separates proved implementation properties, deterministic contract cases, open empirical hypotheses, and claims that are false for this release.

Table 1: Epistemic status of the formal claims.

Statement	Status	Evidence required
Field values remain in $[0, 10.0]$	Proved implementation invariant	Recurrence and saturation tests
A passive trace decays linearly and disappears in finite ticks	Proved implementation invariant	Exact tick tests
More same-target RISK or FAILURE pressure cannot increase the ordinary score	Proved arithmetic invariant	Monotonicity and paired integration tests
One canonical failed outcome moves the paired lower-tier case from 0.875/EXECUTE to 0.725/HOLD	Verified deterministic case	Real subsystem integration test
Gate score lies in $[0.0, 1.0]$	Proved arithmetic invariant	Component ranges and clip
Trust lies in $[0.0, 1.0]$	Proved implementation invariant	Clipped update
The gate lowers production harm	Open empirical hypothesis	Linked, externally attested, representative trials
HOLD improves decisions	Open value-of-information hypothesis	Revision-cost and outcome study
The ecology converges or is optimal	Not established	A specified stochastic model and external validation
Published trust is differentially private	False for the current release	A randomized mechanism and privacy accounting

The result is intentionally narrower than an abstract theory of agent societies. It is a checkable contract for this implementation and a map of what must be measured before stronger claims are warranted.

3 Implementation Method and Control-Plane Semantics

3.1 Colony Kernel Architecture

3.1.1 Overview of the 8 Subsystems

The Colony Control Plane is realised as a single Python package (`codomyrmex.colony_kernel`) with shared value objects and enumerations in `models.py`, canonical subsystem implementations in standalone modules, and the `ColonyKernel` integration class orchestrating their lifecycle. The design keeps communication explicit: subsystems exchange typed models, while the kernel owns cross-subsystem sequencing and state transitions.

Table 2 enumerates the 8 subsystem roles used by the control plane.

Table 2: Colony Control Plane subsystem overview.

Subsystem	Primary Module	Responsibility
PheromoneStore	`pheromone_store.py`	Stores typed location signals; deposits, queries, reinforces, and subtractively decays traces.
ResourceLedger	`resource_ledger.py`	Checks and consumes 7-dimensional, period-scoped resource budgets.
ActuationGate	`actuation_gate.py`	Combines budget approval, effective local hazard, trust, and completeness into EXECUTE / HOLD / REFUSE.
ConsequenceMemory	`consequence_memory.py`	Stores reported outcomes and profiles; computes trust deltas and recent-failure counts. Default storage is in memory.
RoleAdapter	`role_adapter.py`	Infers role labels from trust and proposal count; only SANDBOX has role-specific gate behavior.
PruningDaemon	`pruning_daemon.py`	Nominates stale or duplicate modules; mutation is a separate explicit API.
FalsificationWorker	`falsification/`	Runs 11 deterministic checks across 10 attack-vector categories and deposits finding signals.
ColonyKernel	`kernel.py`	Owns subsystem instances and sequences the high-level proposal, outcome, status, pruning, and tick APIs.

Unless a table or experiment explicitly says otherwise, the numeric weights, thresholds, decay settings, and effort profiles described below are current implementation defaults and example/initial values. They are configurable policy parameters, not calibrated universal constants; tuning them requires rerunning the focused contract tests and regenerating the dependent evidence bundle.

The integration entry point is `ColonyKernel`, instantiated once by the default MCP adapter. Its high-level methods coordinate the components, while lower-level classes remain importable for standalone use and testing.

Figure 3 illustrates the control-plane topology: `ColonyKernel` at the centre, each of the 7 operational subsystem classes at the leaves, all sharing the `models.py` value-object contract.

ColonyKernel occupies the central hub. Radial spokes connect it bidirectionally to the labelled Pheromone Store, Resource Ledger, Actuation Gate, Consequence Memory, Role Adapter, Pruning Daemon, and Falsification Worker nodes. Subtitles state each node's function, and arrowheads show information flow. The topology describes software ownership, not hosts, latency, or throughput.

3.2 Pheromone Gradients (PheromoneStore)

The `PheromoneStore` encapsulates process-local environmental memory in the tradition of stigmergic coordination (Grassé 1959; Dorigo and Stützle 2004). Rather than requiring direct agent-to-agent messages, the running kernel writes typed traces into a shared in-memory `TraceField` and reads them during gate evaluation. Unreinforced traces lose a fixed amount per tick and are removed at zero. This is an engineering analogy to stigmergy, not a claim that the store implements biological chemistry or variational free-energy minimisation.

Signal types. The store recognises 6 signal types (`SignalType` enum), each with distinct ecological meaning:

Table 3 lists the signal classes and their effect on the gate.

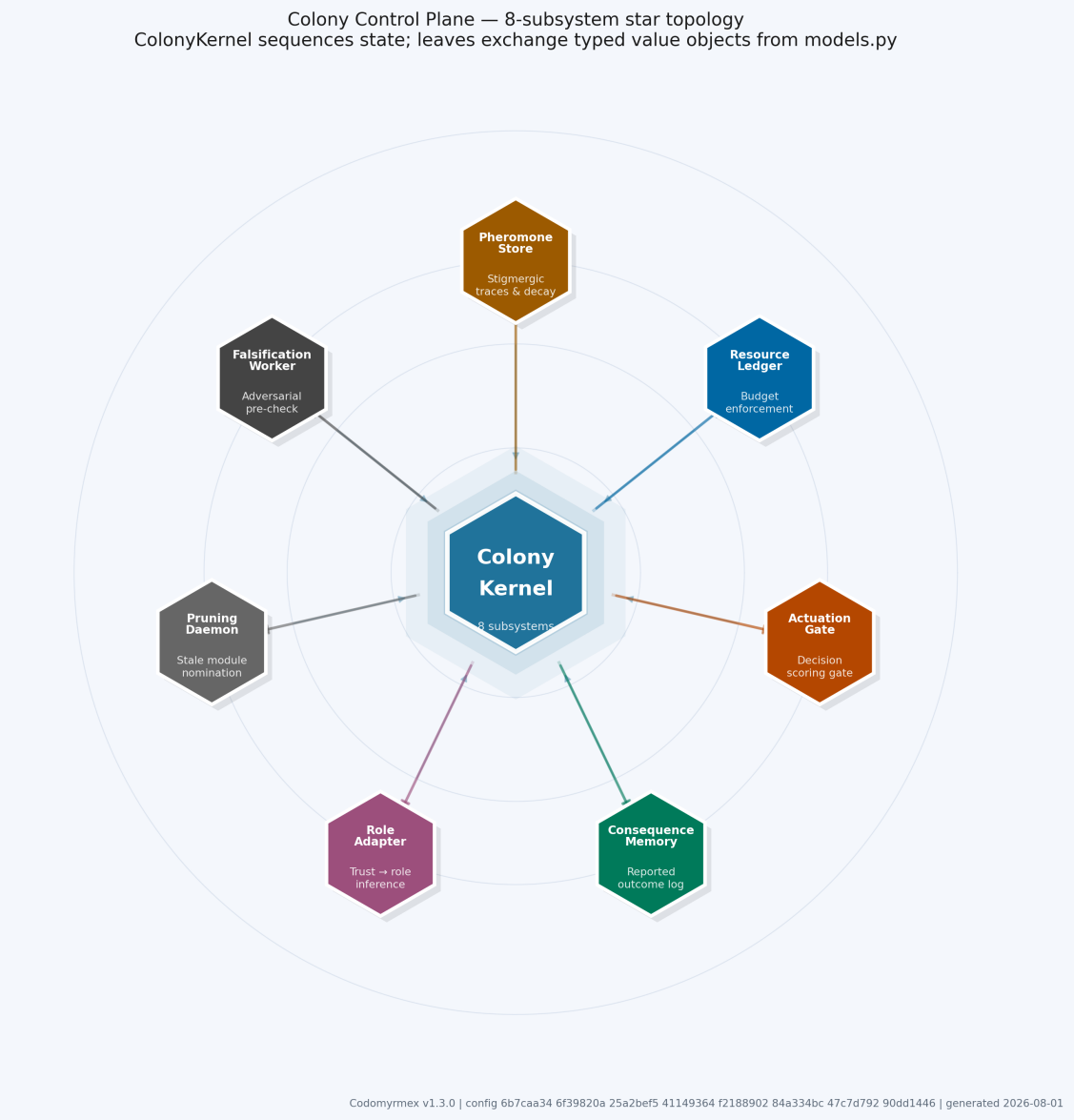

Figure 3: Colony Control Plane topology. ColonyKernel owns subsystem lifecycle and sequencing; the live integration object supplies the operational leaf-node count, and the nodes exchange typed value objects from the shared models.py contract. Node labels, radial position, and arrow direction identify each functional role; colour is a secondary cue. The diagram shows ownership and information flow, not deployment topology or performance.

Table 3: Pheromone signal types and gate effects.

Signal Type	Ecological Analogue	Decay Class	Effect on Gate
`FAILURE`	Trail avoidance pheromone	`FAST`	Records failed outcome reports; the gate scores the maximum of local FAILURE and RISK pressure.
`SUCCESS`	Trail amplification pheromone	`SLOW`	Reinforced on passing outcomes and retained as positive local history; it does not reduce `risk_ok` directly.
`RISK`	Caution marker	`FAST`	Prospective concern channel; combined with FAILURE through a maximum when computing local hazard pressure.
`NEED`	Resource request	`NORMAL`	Broadcast by agents requiring attention at a location; does not feed gate pressure directly.
`DEPENDENCY`	Usage trace	`SLOW`	Deposited by `record_outcome` to signal active module consumption; a strength ≥ 2.0 vetoes `PruningDaemon` nomination.
`HUMAN_PRIORITY`	Operator-injected signal	`SLOW`	Receives the HUMAN source multiplier and remains visible to diagnostics; it is not a gate override.

Decay rates. Each signal is assigned one of 3 `DecayRate` classes. The enum value is a multiplier m_k applied to the base subtraction $\epsilon_0 = 0.1$ strength units per tick. The runtime update is linear and floored at zero, as formalized in Equation 2.

Table 4 records unit-trace behavior for each decay class.

Table 4: Subtractive pheromone decay classes for a unit-strength trace.

Class	Source multiplier	Subtraction ϵ_0 multiplier	Unit trace after one tick	Unit-trace extinction
`FAST`	3.0	0.3/tick	70%	4 discrete ticks
`NORMAL`	1.0	0.1/tick	90%	10 discrete ticks
`SLOW`	0.2	0.02/tick	98%	50 discrete ticks

Extinction scales with deposited strength and reinforcement. The table therefore does not assign a universal half-life to a class; it gives a reproducible unit-trace example. FAST warnings clear quickly, while SLOW success, dependency, and priority traces retain more history.

Trust multipliers by source. Not all sources carry equal epistemic weight. The `PheromoneStore` scales a signal's effective initial strength by the depositing source's trust multiplier before writing to the `TraceField`:

The effective-strength calculation in Equation 15 determines the initial trace written to the field.

$$d_{j,t} = \text{signal.strength} \times \text{source\_multiplier} \times \text{trust\_factor} \tag{15}$$

where `source_multiplier` is HUMAN×2.0, TEST×1.5, SECURITY×1.5, AGENT×1.0, or RUNTIME×1.0. The optional `trust_factor` is supplied only on selected kernel deposits; many runtime and test deposits use its neutral default.

Compound key addressing. Each pheromone occupies a unique position in the `TraceField` identified by the compound key `"{location}:{signal_type.value}"`. This allows FAILURE and SUCCESS signals at the same module path to be tracked and decayed independently, preserving the sign and type of the colony's accumulated evidence.

The generated unit-trace trajectories and their assumptions appear in Figure 1.

3.3 Resource Budget (ResourceLedger)

The `ResourceLedger` enforces a multi-dimensional budget envelope over the colony's resource consumption (Bonabeau et al. 1999). Rather than tracking a single scalar cost, it maintains 7 independent dimensions of `ResourceCost`:

Table 5 lists the dimensions enforced by the ledger.

Table 5: Resource budget dimensions enforced by `ResourceLedger`.

Dimension	Type	Description
`llm_calls`	`int`	Number of LLM API invocations.
`runtime_seconds`	`float`	Wall-clock execution time.
`risk_level`	`float` $\in$ `[0.0,1.0]`	Aggregate risk fraction of the action.
`human_attention_minutes`	`float`	Estimated operator review time.
`merge_risk`	`float` $\in$ `[0.0,1.0]`	Probability of merge conflicts or integration failures.
`doc_debt`	`float`	Documentation gap accumulation score.
`security_exposure`	`float` $\in$ `[0.0,1.0]`	Estimated security surface increase.

The ledger's `can_afford` method returns an `(approved, reason)` pair after checking whether the proposed estimate, added to usage in the current reset period, would breach any dimension. A false approval bypasses ordinary scoring (`gate_score = 0.0`) because budget failure is an early return rather than a score penalty. The final decision depends on the calling mode. In standalone mode, `ActuationGate.evaluate(proposal, profile)` performs the ledger check and returns `REFUSE`; in integrated mode, `ColonyKernel` supplies `budget_approved=False` and the gate returns `HOLD` so a caller may retry after the period resets. On a submitted outcome report, `consume` uses `outcome["cost"]` when it is a valid resource-cost mapping and otherwise falls back to the proposal estimate. The accumulator resets when elapsed time crosses the configured `period_seconds`; this is a fixed-period reset from the last start or reset, not a continuously sliding window.

3.4 Actuation Gate

The actuation gate is the colony's central advisory decision layer: it aggregates signals from the resource ledger, pheromone field, agent trust store, and proposal completeness into a scalar gate score, then routes that score to one of 3 decisions.

Parameter status. The numeric values shown here are current implementation defaults or illustrative initial settings, not universal constants, fitted parameters, or empirical estimates. They are configurable and can be tuned or replaced through the corresponding runtime policy, configuration, or presentation inputs; after a change, regenerate the tests, tables, figures, and manuscript before interpreting the result. In particular, the weights, thresholds, trust deltas, decay amounts, and presentation ranges below describe the current release snapshot. "Configured" therefore means "declared by the current runtime/configuration contract," not "validated as optimal for every deployment." The numerical details in this subsection are example/initial values that can be tuned through their owning runtime or presentation configuration. Any tuning should rerun the contract suite and regenerate the dependent tables, figures, captions, and provenance before a new interpretation is made.

3.4.1 Gate Scoring Formula

The gate score g is a weighted linear combination of 4 normalised components (Equation 16):

$$g = 0.3b + 0.3\rho(h) + 0.25u + 0.15c \tag{16}$$

clamped to $[0.0, 1.0]$ after summation: $g \leftarrow \max(0.0,\ \min(1.0,\ g))$.

The weights sum to 1.0. In the implementation, the code components `budget_ok`, `risk_ok`, `trust_ok`, and `completeness` are the glossary quantities b, $\rho(h)$, u, and c, respectively. Each component is described below.

`budget_ok` ($w = 0.3$) is binary resource headroom. A false approval bypasses ordinary scoring; each proposal receives a fresh pre-check.

`risk_ok` ($w = 0.3$): The gate senses both RISK and FAILURE at the proposal target and defines effective local hazard as their maximum. It maps that pressure through two module-level constants:

Table 6 defines the discrete mapping from effective local hazard pressure to the normalised score component.

Table 6: Local hazard-pressure mapping used by the gate.

`max(RISK, FAILURE)` pressure	`risk_ok`
≥ 6.0 (`_HIGH_RISK_THRESHOLD`)	0.0
≥ 3.0 (`_MEDIUM_RISK_THRESHOLD`)	0.5
< 3.0	1.0

A location accumulating either prospective RISK findings or reported failed outcomes can therefore lose risk credit until the relevant trace decays. SUCCESS is retained as a separate diagnostic channel and does not cancel the maximum.

`trust_ok` ($w = 0.25$): Agent trust score normalised to gate range. The gate reads `AgentTrustProfile.trust_score` and maps it as follows:

Table 7 gives the hard floor and two scoring tiers for trust.

Table 7: Trust-score mapping used by the actuation gate.

`trust_score`	`trust_ok`
≥ 0.6	1.0
$0.3 \leq$ trust_score < 0.6	0.5
< 0.3	hard REFUSE (early return, `gate_score = 0.0`)

A new agent starts at `trust_score = 0.1`; profiles between the hard floor and full-credit threshold receive 0.5 trust credit.

Trust penalty. When `ConsequenceMemory` is available and the agent's recent-failure count reaches 3, the gate applies:

$$u \leftarrow \max(0.0,\ u - 0.25) \tag{17}$$

This decrement reduces `trust_ok` — the gate's normalised trust contribution for this evaluation only. It does not modify the agent's persistent `trust_score` stored in `ConsequenceMemory`. The agent's durable trust record is updated separately on each `record_outcome` call; the gate penalty is a single-evaluation correction that makes the gate more conservative while an agent is on a losing streak, without permanently penalising the agent's history.

`completeness` ($w = 0.15$): Evidence mass of the proposal. The gate inspects 3 fields: `rollback_plan`, `evidence`, and `expected_outcome`.

The proposal-completeness expression in Equation 18 supplies the ordinary score component for incomplete proposals.

$$c = \max(0.0,\ 1.0 - |\text{missing}| \times 0.35) \tag{18}$$

The rendered result tables compute all discrete completeness values from that expression.

3.4.2 Decision Thresholds

The gate maps the numeric score to a ternary verdict:

Table 8 gives the three threshold bands.

Table 8: Actuation-gate decision thresholds.

Score Range	Decision	Effect
$g \geq 0.75$	EXECUTE	Advisory approval returned; the kernel does not perform or enforce the action.
$0.5 \leq g < 0.75$	HOLD	Revision or recovery evidence returned; the kernel does not maintain a requeue.
$g < 0.5$	REFUSE	Refusal returned and, in the integrated path, FAILURE pressure deposited at the target.

3.4.3 Hard Overrides

Four safety conditions bypass the numeric score entirely, evaluated in order before gate scoring begins:

1. Budget failure. A `False` return from `ResourceLedger.can_afford` causes an immediate hard override with `gate_score = 0.0`. Standalone gate evaluation returns `REFUSE`; kernel/caller-supplied budget failure returns `HOLD` for requeue after the budget period resets. No further evaluation occurs.
2. SANDBOX role. Agents with role `SANDBOX` always receive REFUSE regardless of trust score, pheromone state, or proposal quality. The entry label is held until at least 3 proposals and the live promotion threshold are satisfied.
3. Trust floor. A `trust_score < 0.3` triggers an early REFUSE with `gate_score = 0.0`.
4. Critical falsification. Any `CRITICAL` finding from the `FalsificationWorker` triggers an immediate REFUSE with the finding's remediation attached to `GateResult.required_evidence`.

3.4.4 Generated policy cases

Table 23 in Section 5 is generated from the live weights, thresholds, tier mappings, and missing-field penalty. This avoids maintaining a second, hand-calculated worked example in prose.

3.5 Consequence Memory (SQLite-backed)

The `ConsequenceMemory` subsystem stores each reported `ConsequenceRecord` and the derived agent profile. It uses SQLite WAL mode when configured with a file path. The kernel and MCP defaults use `:memory:`, so records survive only for the process lifetime unless an operator supplies a persistent database path. On the ordinary MCP path, the outcome endpoint does not attest reports against a prior EXECUTE record, so the database is an audit log of submitted outcomes rather than independent ground truth. Optional kernel attestation provides a separate linked lifecycle path; required mode rejects the ordinary outcome method and requires a proposal, EXECUTE verdict, authorization, execution receipt, and signed outcome event. Neither mode independently observes external actuation.

The schema comprises three tables: `consequences` (one row per submitted consequence report), `agent_profiles` (one row per agent, containing current trust state and role), and `consequence_history` (a chronological sequence of consequence IDs per agent, capped at the most recent 200 rows).

Trust delta computation. When a `ConsequenceRecord` is persisted with `trust_delta == 0.0`, the memory computes it from outcome fields:

The durable trust update is computed by Equation 19. Notation follows Section 12; Δ_n is a trust increment, not a field signal.

$$\Delta_n = \delta_{\text{test}}(n) + \delta_{\text{repair}}\, 1_{\text{repair}}(n) + f_n \cdot \delta_{\text{human}} \tag{19}$$

where $\delta_{\text{test}}(n) = 0.04$ if tests passed else -0.08; $\delta_{\text{repair}} = -0.05$ is applied when $1_{\text{repair}}(n) = 1$; $f_n \in [-1.0, 1.0]$ is the parsed human feedback score and $\delta_{\text{human}} = 0.03$. The increment is clamped to keep `trust_score` within $[0.0, 1.0]$.

`recent_failures()` and gate coupling. The `ConsequenceMemory` exposes a `recent_failures(agent_id, window=10)` method that the `ActuationGate` queries when a consequence-memory reference is provided. When failed reports among the agent's 10 most recent consequence records reach 3 or more, the gate applies an additional -0.25 decrement to `trust_ok`—the

normalized trust contribution for that evaluation—not to the persistent `trust_score`. The durable score is updated separately from the submitted test and human-feedback fields on `record_outcome`. This is a report-dependent feedback loop, not an independently observed performance measure.

3.6 Role Adaptation

Agent roles are not assigned at registration — they are inferred deterministically by `RoleAdapter.infer_role` from a trust profile's `trust_score` and `total_proposals` count. New agents enter at trust 0.1 (the `SANDBOX` score from `AgentTrustProfile` default). The `AgentRole` enum defines exactly 5 roles:

Table 9 summarises the kernel-facing role ladder.

Table 9: Kernel role ladder and intended specializations.

Role label	Trust Required	Minimum Proposals	Intended specialization
`SANDBOX`	Any	< 3	Entry/quarantine label; the current gate refuses every proposal, including read-only proposals.
`REPAIR_ANT`	≥ 0.2	≥ 3	Patch, test-fix, documentation update.
`MEMORY_ANT`	≥ 0.35	≥ 3	Archive, index, summarise.
`DISPATCHER`	≥ 0.5	≥ 3	Delegate, coordinate, route tasks.
`GUARD_ANT`	≥ 0.7	≥ 3	Security review, gate audit, archive authority.

Promotion is governed by the `RoleAdapter.infer_role` ladder shown in the table. An agent must first accumulate at least 3 total proposals; before that minimum, it remains `SANDBOX` regardless of trust score. Once the proposal minimum is met, the four thresholds are generated directly from the live adapter. The `trust_promote_threshold` runtime projection records the first promotion floor (0.2); it is not a separate general threshold.

Role inference runs on proposal and outcome cycles. The live gate enforces the SANDBOX override but does not implement a per-action permission matrix for the four higher role labels; those labels currently express trust tiers and intended specializations. Outcome recording persists a changed role. The transition out of SANDBOX depends on externally recorded outcomes, which are not yet linked to prior authorized execution.

`RoleAdapter` also exposes a standalone `assign_role()` API that is not called by the kernel's proposal path. This specialization-based path applies higher trust thresholds (0.8 for REPAIR_ANT and MEMORY_ANT, 0.85 for GUARD_ANT) and matches action types (e.g. `test_fix`, `doc_write`, `security_scan`) to role labels. It is a separate interface for callers that want action-aware role assignment; the kernel path uses only the trust-and-proposal-count ladder above.

3.7 Pruning Daemon (Death and Pruning)

The `PruningDaemon` performs periodic ecological thinning of the colony's module registry, identifying stale or redundant components before they accumulate into structural debt. It is the colony's analogue of the biological apoptosis mechanism (Bonabeau et al. 1999).

The daemon operates by scanning a `module_registry` dict (mapping dotted module paths to usage metadata) and classifying each entry according to four confidence tiers:

Table 10 records the nomination confidence tiers.

Table 10: Pruning-daemon candidate confidence tiers.

Condition	Confidence	Reason Tag
`call_count == 0` and `last_used == 0.0`	0.9	`never used since registration`
Duplicate of another module	0.85	`duplicate of <surviving_module>`
Zero calls, last used > 30 days ago	0.7	`no calls; last used N days ago`
Low call count (< 5), last used > 30 days	0.5	`low usage (N calls); last used N days ago`

Before classifying any module, the daemon checks the colony's `PheromoneStore` for a `DEPENDENCY` signal at that path. A pheromone strength ≥ 2.0 indicates the module is actively consumed; the candidate is suppressed regardless of call-count metadata. Pheromones act as a veto on the daemon's statistical inference: a module that is genuinely active can receive DEPENDENCY deposits from reported outcomes and thereby suppress nomination.

The normal scan path returns `PruningCandidate` reports sorted by confidence. Explicit non-dry-run pruning APIs can archive candidates, so safe deployment must keep nomination and actuation separately authorized. The daemon does not autonomously schedule deletion, and stale-document detection remains incomplete.

3.8 Falsification Worker

The `FalsificationWorker` is the colony's deterministic adversarial-review component. It runs 11 heuristic checks grouped into the 10 `AttackVector` categories, without an LLM call. The design is motivated by falsifiability (Popper 2002), but each check is a software heuristic rather than a scientific test of truth.

Attack vector taxonomy. The 10 attack vectors are defined in the `AttackVector` enum. Severity weights follow `_SEVERITY_RANK` (LOW=1, MEDIUM=2, HIGH=3, CRITICAL=4):

Table 11 lists the canonical adversarial vectors.

Table 11: Falsification-worker attack vector taxonomy.

Vector	Typical Severity	Weight	Description
`NO_ROLLBACK`	HIGH	3	Plan lacks a concrete, non-placeholder rollback path.
`NO_TEST_VALUE`	HIGH	3	Plan includes no automated test coverage assertion.
`SCOPE_CREEP`	MEDIUM–HIGH	2–3	Plan's scope is vague or reaches beyond its stated target.
`FALSE_METRIC`	LOW–MEDIUM	1–2	Expected outcome is absent, tautological, or non-falsifiable.
`CIRCULAR_ARCHITECTURE`	MEDIUM–HIGH	2–3	Proposed changes introduce circular import dependencies.
`DEPENDENCY_RISK`	MEDIUM	2	Plan introduces at least 3 unvetted external packages.
`SECURITY_RISK`	HIGH	3	Plan touches security-sensitive surface without a review annotation.
`OVER_BROAD_MODULE`	MEDIUM	2	Module rationale uses at least 5 responsibility verbs.
`HIDDEN_MAINTENANCE_COST`	MEDIUM	2	Plan does not account for long-term upkeep burden.

Vector	Typical Severity	Weight	Description
PREMATURE_ABSTRACTION	LOW	1	Plan introduces a generic abstraction without demonstrated need.

The CIRCULAR_ARCHITECTURE check has repository-aware and plan-only modes. With a repo_root, it parses Python imports and uses depth-first traversal with a recursion stack to find cycles. Without a repository root, it inspects declared dependencies for self-reference and parent–child patterns.

Pheromone deposits are best-effort. HIGH and CRITICAL findings deposit FAILURE; MEDIUM findings deposit RISK; LOW findings deposit neither. The kernel may add a further RISK trace for non-refused proposals with MEDIUM-or-higher findings.

Each check returns a FalsificationFinding with fields for the attacked claim, vector, severity, evidence, and remediation. The evaluate_plan() method runs all 11 checks and returns a FalsificationReport with an overall verdict:

- PASS: zero findings.
- CONDITIONAL: one or more findings, all LOW or MEDIUM.
- FAIL: any finding reaches HIGH or CRITICAL (rank ≥ 3).

The report verdict and gate decision are distinct. Only a CRITICAL finding is a gate hard override. HIGH findings appear in the reason and pheromone field but do not by themselves force REFUSE; ordinary score components determine the result.

Figure 4 shows all 10 canonical attack vectors ranked by maximum severity weight. The current deterministic checks top out at HIGH severity; the CRITICAL class remains part of the actuation-gate override contract.

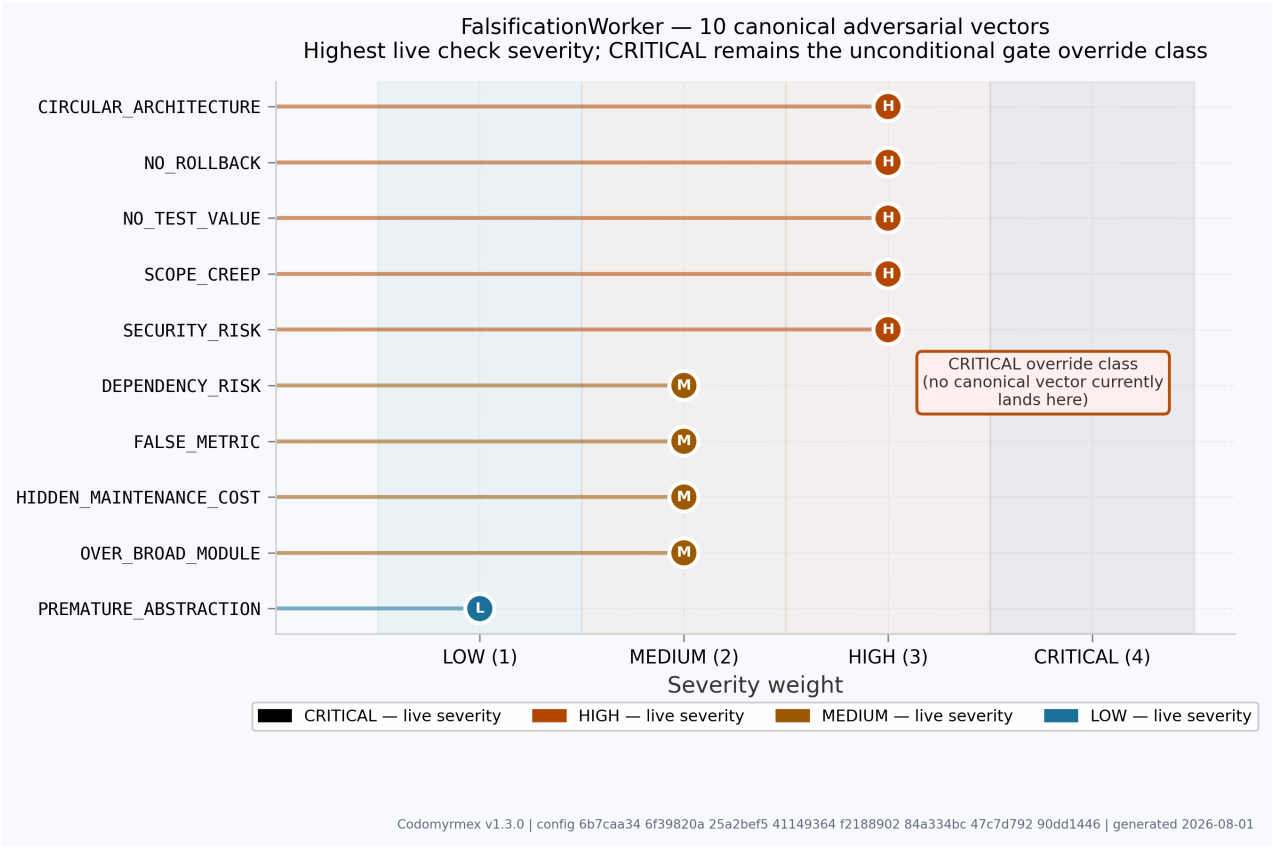

Figure 4: Falsification categories ordered by the highest severity emitted by their live heuristic checks. Horizontal position, a printed severity initial, and colour redundantly encode severity class; one category may be served by more than one check. The empty CRITICAL band denotes the only finding class that hard-refuses before ordinary scoring. The chart is a code-taxonomy visualization, not a measured detection-rate comparison.

Each row names one live adversarial vector. A line ends at that vector's highest emitted severity rank, and the marker contains the severity initial. Rows are sorted from highest to lowest severity. The CRITICAL column is explicitly labelled even when empty because that class alone hard-refuses before scoring. The plot shows code taxonomy, not prevalence or detection rate.

3.9 Supporting Modules and Research Adapters

Beyond the eight primary subsystems, the Colony Kernel package includes several supporting modules that strengthen verification, configuration, and research extensibility. These modules are part of the checked-in codebase and are exercised by the test suite, but they are not counted among the eight primary control-plane subsystems because they serve verification, configuration, or research roles rather than runtime actuation.

3.9.1 Configuration loading (**`config_loader.py`**)

`config_loader.py` loads YAML-backed configuration from `config/colony_kernel/` (`kernel.yaml`, `roles.yaml`, `decay_rates.yaml`). It provides structured defaults for gate thresholds, trust boundaries, role promotion criteria, and pheromone evaporation rates. The manuscript variable generator reads the same live constants from source, so the configuration file and the code are kept in sync by the `float_mirrors` and `exact_mirrors` validation in `variables.py`.

3.9.2 Invariant predicates (**`invariants.py`**)

`invariants.py` contains executable design-by-contract predicates that check selected properties of the runtime state: gate-weight conservation, trust-score range, pheromone-strength bounds, role-ladder monotonicity, and enum-value separation. The function `all_invariants_hold()` runs all checks against live constants; these predicates are distinct from the SMT obligations in `formal.py` because they execute against runtime values rather than symbolic expressions.

3.9.3 Reference gate (**`reference.py`**)

`reference.py` provides an independent deterministic reimplementation of the gate-scoring arithmetic with parameters that mirror the live gate constants (execute threshold, hold threshold, trust hard floor, etc.). It is used for differential testing: the reference gate and the production `ActuationGate` should produce identical scores for identical inputs. This module is mentioned in Section 11.3 as part of the refinement-test bridge.

3.9.4 Formal bridge (**`formal.py`**)

`formal.py` defines runtime obligations and an optional solver-neutral result bridge. The `KernelFormalSnapshot` dataclass captures a bounded kernel state; `runtime_obligations()` evaluates selected properties (weight conservation, trust range, pressure monotonicity, unrelated-target locality, authorized outcome linkage) against that snapshot. The optional Z3 backend in `formal_verification/z3_bridge.py` translates a subset of these obligations into solver expressions, returning `proved`, `refuted`, `unknown`, `timeout`, or `unavailable` (when Z3 is not installed). The bridge proves or refutes only the encoded bounded obligations; it does not establish whole-program refinement or production safety.

3.9.5 Attestation ledger (**`attestation.py`**)

`attestation.py` implements a versioned, hash-chained execution ledger that binds proposal, verdict, authorization, execution receipt, outcome, rejection, and error events under a configurable signer. The default `HMACSigner` uses constant-time comparison; an optional `Ed25519Signer` uses the `cryptography` library with a delayed import. The ledger supports optional and required attestation modes: in required mode, unlinked outcomes and duplicate nonces are rejected. This ledger is not automatically inserted into the default caller-reported MCP path, so the release does not claim that every outcome report is authenticated. The ledger is exercised by focused tests for replay, omission, duplication, nonce reuse, and unauthorized relinking.

The `AttestationLedger` is an optional additive component, not one of the 8 core subsystems. `ColonyKernelConfig` accepts an `attestation_mode` of `disabled` (default), `optional`, or `required`. When disabled, `attestation_ledger` is `None` and the kernel operates without the ledger; when `optional` or `required`, the ledger is instantiated and the kernel records proposal, verdict, authorization, receipt, and outcome events. In `required` mode, `record_outcome` rejects outcomes that lack a prior authorized execution receipt.

3.9.6 Replay harness (**replay.py**)

`replay.py` provides the fixed-input paired-locality replay used to generate the deterministic fixture evidence reported in Section 5.2. The function `run_paired_locality_replay()` constructs a real `ColonyKernel` instance, runs the paired scenario, and returns both semantic and file digests. The replay is repeated for semantic equality; it is not an attestation, causal estimate, or external workload result.

3.9.7 Research subpackage (**research/**)

The `research/` subpackage contains offline-first adapters that are not called by the production gate:

- `benchmark.py` — `run_paired_benchmark()` executes deterministic synthetic baseline and mediated traces and computes descriptive safety, utility, refusal, and trace metrics. The paired cases are synthetic; they demonstrate analysis plumbing, not calibration or generalization.
- `metrics.py` — descriptive log loss, Brier score, calibration error, selective risk, and bootstrap confidence intervals.
- `persistent_store.py` — SQLite WAL persistence adapter with crash-injection boundaries for durability testing.
- `probabilistic.py` — `KernelProbabilisticAdapter` declares an observation model, latent-state space, priors, likelihoods, transitions, preferences, and policy horizon. It is research plumbing, not an integrated Active Inference controller.
- `schemas.py` — versioned cases, traces, manifests, and leakage reports.

These adapters preserve the boundary between deterministic runtime contracts and research interpretations. They are exercised by focused tests but do not relabel deterministic gate scores as probabilistic outputs.

3.10 The Pressure Loop

The feedback path is composed from separate public operations. `propose_action` returns a decision but does not actuate. With attestation disabled or optional, `record_outcome` accepts a later caller report without requiring a matching authorization. Optional mode also exposes a linked path; required mode rejects the ordinary method and admits only `record_attested_outcome` after authorization and an execution receipt. The pseudocode separates those contracts.

Algorithm 1: Colony Kernel Pressure Loop and Attestation Boundary

```
Input:  ActionProposal p from any agent
Output: GateResult verdict; side effects on PheromoneStore,
        ResourceLedger, ConsequenceMemory

PROPOSE(p):
         findings ← FalsificationWorker.analyze(p)
             [11 checks; HIGH+ deposits FAILURE, MEDIUM deposits RISK]
         budget_approved ← ResourceLedger.check_budget(p.budget_estimate)
         profile ← ConsequenceMemory.get_profile(p.agent_id)
         profile.total_proposals ← profile.total_proposals + 1
         ConsequenceMemory.save_profile(profile)
         RoleAdapter.update(profile)
         witness ← ActuationGate.witness_state(p)
         hazard ← max(witness.RISK, witness.FAILURE)
         if budget_approved is false:
             decision ← HOLD; gate_score ← 0.0; goto POST_GATE
         if profile.role is SANDBOX or profile.trust_score < 0.3:
             decision ← REFUSE; gate_score ← 0.0; goto POST_GATE
         if any CRITICAL finding:
             decision ← REFUSE; gate_score ← 0.0; goto POST_GATE
         trust_ok ← 1.0 if profile.trust_score >= 0.6 else 0.5
         if ConsequenceMemory.recent_failures(p.agent_id) >= 3:
             trust_ok ← max(0.0, trust_ok − 0.25)
         hazard_ok ← 0.0 if hazard >= 6.0
                   | 0.5 if hazard >= 3.0
                   | 1.0 otherwise
         missing ← [f for f in [rollback_plan, evidence, expected_outcome] if absent]
```

```
        completeness ← max(0.0, 1.0 − len(missing) ∗ 0.35) if missing else 1.0
        gate_score ← 0.3 + hazard_ok ∗ 0.3
                    + trust_ok ∗ 0.25 + completeness ∗ 0.15
        gate_score ← max(0.0, min(1.0, gate_score))
        decision ← EXECUTE if gate_score >= 0.75
                 | HOLD   if gate_score >= 0.5
                 | REFUSE otherwise
POST_GATE:
        if decision is REFUSE: deposit FAILURE at p.target
        return GateResult(decision, gate_score, reason, required_evidence)

ORDINARY PATH: CALLER MAY ACTUATE AFTER EXECUTE; THE KERNEL DOES NOT ENFORCE THIS STEP.

ORDINARY_REPORT(p, outcome, tests_passed):
        [Available only when attestation is disabled or optional]
        [No consumed authorization or duplicate-report check]
        record ← ConsequenceMemory.record(reported consequence)
        ResourceLedger.consume(outcome.cost or p.budget_estimate)
        if tests_passed and no repair: deposit/reinforce SUCCESS
        if tests failed: deposit FAST FAILURE
        deposit SLOW DEPENDENCY at p.target
        profile ← ConsequenceMemory.get_profile(p.agent_id)
        if RoleAdapter.update(profile): ConsequenceMemory.save_profile(profile)
        return record

ATTESTED_REPORT(p, outcome, tests_passed):
        require an attested EXECUTE verdict for p
        authorization ← authorize_execution(p.proposal_id)
        caller actuates only after authorization
        execution ← record_execution_receipt(authorization, caller receipt)
        append signed outcome event linked to execution
        apply the same consequence, budget, trust, role, and signal updates
        [Required mode rejects ORDINARY_REPORT; local linkage does not observe actuation]

TICK(): subtract each trace's per-tick evaporation and check budget rollover
PRUNING_REPORT(registry): scan and return candidates; do not archive by default
```

The loop is a deterministic feedback pattern produced by explicit kernel calls: agents propose; callers optionally execute approved work; callers separately report outcomes; trust and local traces then affect later evaluations. Failed outcomes raise same-target hazard pressure and lower the reporting agent's trust. This can move later proposals from EXECUTE to HOLD or REFUSE, but the effect is bounded by decay and depends on the accuracy of the submitted outcome. The authenticated path protects linkage and integrity of that submission, not its external truth.

On the ordinary path, the generated weights are budget 0.3, hazard 0.3, trust 0.25, and completeness 0.15. Hard overrides intentionally dominate that arithmetic. The score is therefore a transparent policy composition, not a learned collective judgment or calibrated safety probability.

Figure 5 maps the conceptual data dependencies as a 8-stage loop. It shows which state feeds later decisions; it is not a literal call-order trace, because outcome recording is a separate caller action and falsification runs before scoring.

Eight labelled cards form a clockwise loop. A proposal is falsification-checked, witnessed against state, and evaluated by the gate before a separate caller may actuate it. The caller then reports an outcome, which updates trust, budget, role, and local pheromone pressure for later proposals. Numbers and arrow direction, rather than colour alone, define the sequence.

4 Evaluation Protocol, Configuration, and Reproducibility Inputs

This section separates a proposed external benchmark from the configuration and contract checks that were actually executed for this release. The comparative benchmark has not been run, its baselines are not implemented as released adapters, and no raw trial traces are included. Its purpose here is to make the next empirical test explicit without presenting planned work as evidence.

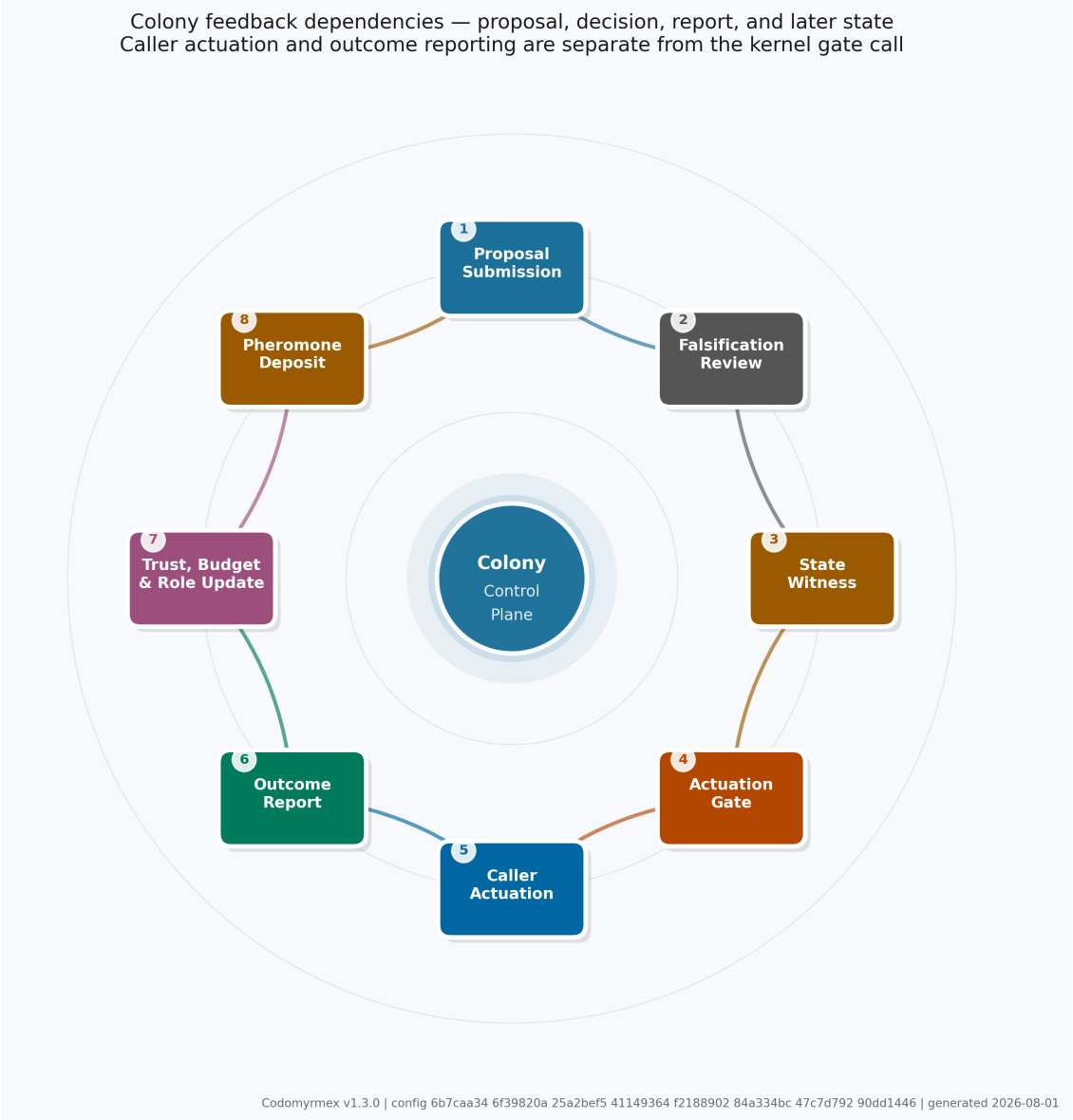

Figure 5: Conceptual Colony Kernel feedback dependencies. Proposal review reads budget, local RISK/FAILURE pressure, role, trust, and completeness; a separate caller may execute an approved action and report its outcome; recording then changes trust, resource usage, and signal traces used on a later proposal. Arrows show feedback dependencies rather than exact runtime call order.

Parameter status. The numeric values shown here are current implementation defaults or illustrative initial settings, not universal constants, fitted parameters, or empirical estimates. They are configurable and can be tuned or replaced through the corresponding runtime policy, configuration, or presentation inputs; after a change, regenerate the tests, tables, figures, and manuscript before interpreting the result. The protocol values reserved in this section are therefore starting points for a preregistered study, not observations or claims about the correct operating point. A study should report every changed value, its authority (runtime policy, configuration, or presentation), and the regenerated artifact hashes.

Table 12 makes the distinction explicit before the proposed trial design and the live runtime configuration are described.

Table 12: Parameter classes, authorities, and evidence interpretation for the current snapshot.

Parameter class	Examples in this release	Interpretation and authority
Runtime policy	Gate weights, thresholds, trust deltas, decay constants	Current code-defined behavior; tune by changing the runtime policy and rerun the contract suite.
Presentation	Plot horizon, grid density, trust slices, checkpoint selection	Reproducible display choices; tune without changing kernel semantics, but regenerate figures and captions.
Proposed protocol	Agent count, workload size, warm-up, trial count, seed, baseline conditions	Initial study-design values; they are not completed experiments or observed sample sizes.

4.1 Evidence-status map

Table 13 is the release's claim-status ledger.

Table 13: Evidence status for the release and proposed study.

Evidence surface	Release status	Permitted inference
Colony Kernel unit/integration suite	Executed during variable generation	Checked deterministic behavior under test inputs
Ruff and ty checks	Executed; fail closed	No scoped lint/type diagnostics in this snapshot
Formula-derived figures and tables	Regenerated from configuration and constants	Arithmetic consequences of the policy
Local authenticated lifecycle ledger	Executed deterministic fixture	Hash/signature/linkage integrity, not external actuation truth
4-condition benchmark	Proposed, not executed	No comparative conclusion
Production deployment study	Absent	No production safety or performance conclusion

4.1.1 Proposed independent variable

The proposed study varies the gate policy while holding the ordered workload and reported outcomes constant:

1. Composite gate — the released budget, local hazard, trust, completeness, role, and falsification policy.
2. Static-trust baseline — a future adapter using only an explicitly specified trust threshold.
3. Budget-only baseline — a future adapter using only resource approval.
4. Always-execute baseline — a future unsafe control that dispatches every proposal.

Conditions 2–4 are design specifications, not checked-in drop-in implementations. Implementing and testing them is a prerequisite to executing the benchmark.

4.1.2 Proposed dependent variables

Table 14 defines the outcomes that a future benchmark must actually record rather than infer from gate decisions alone.

Table 14: Outcomes required for the proposed benchmark.

Variable	Required operational definition
Decision distribution	EXECUTE/HOLD/REFUSE counts per condition and paired workload item
Externally attested failure rate	Failed, independently observed outcomes divided by consumed EXECUTE authorizations
Budget efficiency	Consumed resource units per externally attested successful task
Trust path	Per-agent trust before and after every consumed outcome record
Throughput	Externally attested completed workload items per run
Recovery latency	Proposals/ticks from first HOLD until EXECUTE, REFUSE, or expiry

4.1.3 Falsifiable hypotheses

The study should test, rather than assume:

- H1: the composite gate lowers externally attested failure rate relative to always-execute;
- H2: same-target failure has a larger subsequent gate effect than failure at an unrelated target;
- H3: HOLD provides positive value after accounting for revision cost and delay; and
- H4: safety changes are not explained solely by reduced execution volume.

A lower raw error count is insufficient if a policy simply executes less. Results must report denominators, throughput, and paired workload effects.

4.1.4 Proposed trial structure

The checked-in configuration reserves 20 run indices, 5 agents, 50 workload items, and 10 warm-up ticks. These are planning parameters only. Before use, the study must provide:

- a versioned workload with stable item identifiers;
- deterministic seeds or a declaration that the run is fully deterministic;
- implemented baseline adapters;
- an authorization ledger linking proposal ID, decision, execution, and one consumed outcome;
- an independent outcome oracle or attestation rule;
- raw append-only traces with configuration and commit hashes; and
- an analysis script that regenerates every reported table and confidence interval.

"Independent trial" should be used only when randomness or independently sampled workloads justify independence. Replaying an identical deterministic trace produces replications for reproducibility, not independent statistical samples.

4.1.5 Trust initialization and bootstrap

The default profile begins at trust 0.1 and SANDBOX. The current gate refuses all SANDBOX proposals, including read-only actions, while trust increases only through reported outcomes. Consequently, a trial cannot bootstrap autonomously from this state. A credible benchmark must choose and document one of two approaches:

1. provide a fixed, supervised calibration history before measurement; or
2. implement a restricted SANDBOX action path with local lifecycle authentication and deployment-specific external observation.

The present all-success trajectory is a deterministic contract fixture using submitted outcomes. It must not be described as naturally earned autonomous authority.

4.1.6 Proposed analysis

For paired workload items, report condition-by-item decisions and outcomes before aggregate statistics. Binary externally attested failure outcomes can be analyzed with paired methods or a mixed-effects model that accounts for workload and agent. Ternary gate decisions require a multinomial or ordinal model appropriate to the design. Pre-register exclusions, multiplicity

correction, missing-outcome treatment, and stopping rules. Effect sizes with uncertainty intervals are primary; significance tests are secondary.

4.2 Runtime gate configuration

The integrated kernel runs falsification, checks the budget, loads the agent profile, updates its role label, and evaluates the gate. Table 15 gives the configured routing bands.

Table 15: Ordinary gate routing thresholds.

Outcome	Condition on ordinary score	Runtime meaning
EXECUTE	score ≥ 0.75	Caller may actuate
HOLD	$0.5 \leq$ score < 0.75	Return revision/recovery requirements
REFUSE	score < 0.5	Reject and deposit FAILURE

Budget failure, SANDBOX, trust below 0.3, and CRITICAL falsification are evaluated as early returns. Integrated budget failure yields HOLD; standalone gate use yields REFUSE. The score is a policy value, not a calibrated probability.

4.2.1 Gate weights

Table 16 identifies each ordinary score component and its live runtime input.

Table 16: Ordinary gate score components.

Component	Weight	Runtime input
Budget	0.3	Binary result of the resource pre-check
Local hazard	0.3	Piecewise credit from max(RISK, FAILURE)
Trust	0.25	Tiered trust credit with optional recent-failure penalty
Completeness	0.15	Presence of rollback, evidence, and expected outcome

Weights and thresholds are code-defined policy constants with documentation mirrors in YAML. The current release has not calibrated them against external outcomes.

4.3 Signal-field configuration

6 `SignalType` values share a capped field. Each trace stores its own subtractive evaporation amount at deposit time; Table 17 reports the 3 configured classes.

Table 17: Linear field dynamics for a unit trace without reinforcement.

Class	Subtraction per tick	Unit trace at report tick	Typical channels
FAST	0.3	100% lost	FAILURE, RISK
NORMAL	0.1	generated in Table 25	NEED/default
SLOW	0.02	80% retained	SUCCESS, DEPENDENCY, HUMAN_PRIORITY

The runtime caps field strength at 10.0 and floors at zero. Deposits can have source and trust multipliers, so lifetime scales with effective initial strength. The model does not use exponential half-lives.

4.4 Resource caps

Values in Table 18 are loaded from `config/colony_kernel/kernel.yaml` by the manuscript generator.

Table 18: Selected period-scoped resource budget caps.

Dimension	Cap
LLM calls	500
Runtime seconds	3600
Cumulative risk level	0.8
Security exposure	0.5

The ledger has 7 dimensions in total. The pre-check does not consume resources; `record_outcome` consumes a caller-supplied cost when valid or falls back to the proposal estimate.

4.5 Role labels

Table 19 distinguishes inferred labels from the authorization behavior that is actually enforced.

Table 19: Inferred role labels and current enforcement boundary.

Role	Trust threshold after $\geq$3 records	Live authorization behavior
SANDBOX	entry/default	Every proposal hard-refused
REPAIR_ANT	0.2	No role-specific action matrix beyond leaving SANDBOX
MEMORY_ANT	0.35	No role-specific action matrix
DISPATCHER	0.5	No role-specific action matrix
GUARD_ANT	0.7	No role-specific action matrix

The labels encode a deterministic ladder and intended specializations. They should not be reported as enforced capabilities until the gate validates action type against role.

4.6 Falsification categories

The worker runs 11 checks grouped into these 10 enum categories:

1. `SECURITY_RISK`
2. `NO_ROLLBACK`
3. `NO_TEST_VALUE`
4. `SCOPE_CREEP`
5. `CIRCULAR_ARCHITECTURE`
6. `FALSE_METRIC`
7. `HIDDEN_MAINTENANCE_COST`
8. `DEPENDENCY_RISK`
9. `OVER_BROAD_MODULE`
10. `PREMATURE_ABSTRACTION`

HIGH or CRITICAL findings deposit FAILURE; MEDIUM findings deposit RISK. Only CRITICAL findings are gate hard overrides. The worker's PASS/CONDITIONAL/FAIL report is not itself the gate decision.

4.7 Configuration provenance

3 YAML files under `config/colony_kernel/` document/load kernel budget policy, role metadata, and decay metadata. Runtime authority is mixed:

- `kernel.yaml` is loaded for budget defaults and carries documentation mirrors for some gate constants;
- promotion thresholds are private constants in `role_adapter.py`;
- gate weights, hazard thresholds, completeness penalty, and hard floor are constants in `actuation_gate.py`; and

- per-trace evaporation is derived from `DecayRate` and the base store constant.

The manuscript therefore identifies each value's runtime authority instead of claiming that every parameter is dynamically configurable. The numeric values shown here are current implementation defaults or illustrative initial settings, not universal constants, fitted parameters, or empirical estimates. They are configurable and can be tuned or replaced through the corresponding runtime policy, configuration, or presentation inputs; after a change, regenerate the tests, tables, figures, and manuscript before interpreting the result.

4.8 Software snapshot

Table 20 records the build-facing tools and generated snapshot facts.

Table 20: Software used to generate the release evidence.

Component	Snapshot
Python	3.13.11
Package manager	uv
Test runner	pytest + pytest-cov
Linter	Ruff
Type checker	ty
Config version	1.3.0
Generated	2026-08-01T23:11:17Z

"Generated" is an artifact timestamp. Exact dependency resolution comes from `uv.lock`, not from version ranges in prose.

4.9 Manuscript pipeline

The authoritative route runs from the repository root. Its locked dependency group and explicit paths avoid relying on a template project or an untracked wrapper:

```
uv sync --locked --group docs
uv run --locked python scripts/z_generate_manuscript_variables.py
uv run --locked python scripts/generate_manuscript_figures.py
uv run --locked --group docs python scripts/compile_manuscript.py \
  --manuscript-dir output/manuscript --output-dir output --check --skip-generate
uv run --locked --group docs python scripts/compile_manuscript.py \
  --manuscript-dir output/manuscript --output-dir output \
  --pdf --bookends --pdf-engine lualatex --pdf-standard ua-2 --skip-generate
uv run --locked python scripts/validate_manuscript_integrity.py \
  --require-rendered --online-bibliography
```

Variable generation reruns the scoped tests, branch coverage, Ruff, and ty checks and fails on a non-zero gate. Figure generation reads that snapshot and produces the 18 registered images. Compilation hydrates Markdown, resolves cross-references and citations, renders semantic HTML, hashes an unbookended content PDF, then produces the final distribution PDF with visible first/last bookends in one Pandoc/LaTeX pass. The final PDF hash belongs only in the detached publication manifest, which avoids circular self-hashing.

This route binds internal evidence to `output/pdf/codomyrmex_combined.pdf`, `output/paper-content.pdf`, and `output/paper.html`. It does not prove that the proposed external benchmark has been executed or that the requested PDF standard conforms until an external validator reports success.

5 Executed Contract Results and Evidence Boundary

This section reports only evidence reproducible from the checked-in Colony Kernel. It distinguishes:

- executed quality gates: scoped tests, branch coverage, lint, and static typing;
- deterministic contract cases: exact inputs evaluated by real subsystem instances;
- analytical consequences: arithmetic derived from the implemented score; and
- unexecuted evaluation plans: the comparative benchmark specified in Section 4.1.

No external multi-condition benchmark trace ships with this snapshot. Accordingly, this section reports no production effect size, confidence interval, ecological optimum, or population-level safety rate.

The numeric values shown here are current implementation defaults or illustrative initial settings, not universal constants, fitted parameters, or empirical estimates. They are configurable and can be tuned or replaced through the corresponding runtime policy, configuration, or presentation inputs; after a change, regenerate the tests, tables, figures, and manuscript before interpreting the result. The analytical figures and deterministic fixtures below should be read as reproducible policy probes, not as measurements of a deployed agent population. Their numerical settings are current example/initial values, not universal constants; they can be tuned through the owning code or configuration, with the full evidence chain regenerated after each change.

5.1 Executed quality gates

The fail-closed manuscript generator reruns the Colony Kernel tests and refuses to emit release variables if pytest, Ruff, or ty returns non-zero. Table 21 is therefore a snapshot of executed release gates, not a manually entered scorecard.

Table 21: Executed Colony Kernel quality gates for the rendered snapshot.

Gate	Snapshot result	Interpretation
pytest	819 passing tests	Real tests under `tests/unit/colony_kernel/`
branch coverage	76.6%	Scoped to `src/codomyrmex/colony_kernel`
Ruff	0 findings	Zero required for variable generation
ty	0 diagnostics	Zero required for variable generation

The suite spans 20 test modules and exercises the 8 named subsystems, their integration class, and the 8-tool MCP adapter. Coverage and passing tests establish exercised behavior under the suite; they do not establish that untested faults, adversarial outcome reports, or deployment failures are absent.

The source snapshot contains 7104 nonblank, noncomment lines across 16 top-level Colony Kernel Python files. These counts describe scope, not quality. The exact commands and evidence limitations are recorded in Section 8.

5.2 Paired locality contract

The central implemented claim is evaluated as a paired deterministic case using a real `ColonyKernel`, `PheromoneStore`, `ActuationGate`, and in-memory `ConsequenceMemory`—without mocks.

The fixture uses an authorized agent with trust 0.5, no recent failures, a fully specified proposal, an available budget, and an initially clear target. It then records a failed outcome at that target from another agent and reevaluates both the same target and an unrelated target. Table 22 gives the exact results.

Table 22: Paired same-target inhibition, cross-target isolation, and decay recovery.

Evaluation	Local RISK	Local FAILURE	Effective hazard	Score	Decision
Same target, before failure	0.000	0.000	0.000	0.875	EXECUTE
Same target, after failed outcome	0.000	3.000	3.000	0.725	HOLD
Unrelated target, after failed outcome	0.000	0.000	0.000	0.875	EXECUTE
Same target, after 20 passive ticks	0.000	0.000	0.000	0.875	EXECUTE

The failed outcome deposits nominal FAILURE strength 2.0 from source TEST. The 1.5 source multiplier yields pressure 3.000, changing the hazard credit from full to medium. The resulting score change is

$$\Delta g = 0.3(0.5 - 1) = -0.150. \tag{20}$$

The test also establishes locality: the unrelated target is unchanged. A second test establishes reversibility after passive decay. These facts support a bounded statement: the recorded failure increases friction for later same-location action in the running process. They do not show that the original outcome report was truthful, that the effect persists across restarts, or that the gate reduces real-world harm.

The same fixture is replayed twice by run_paired_locality_replay with fixed proposal identities and no random draws. The generated semantic digest is af96389d d276627c 853eeae9 15a54d9a 536e13ff c4d7b74e 9e9f6277 d8b0aabc; the retained JSON artifact is output/data/colony_kernel_replay.json with file digest d0799e32 5bc2f89a 0d9d8b09 81ddfd8f 9afd2b67 241f922b 7c10c279 7e8bb17b. Repeatability is an assertion of semantic equality for this fixture, not evidence that the caller-reported outcome was attested or that the implementation is deterministic under concurrency, restart, or external workloads.

Figure 6 summarizes the four semantic states and the repeatability assertion in a compact visual form.

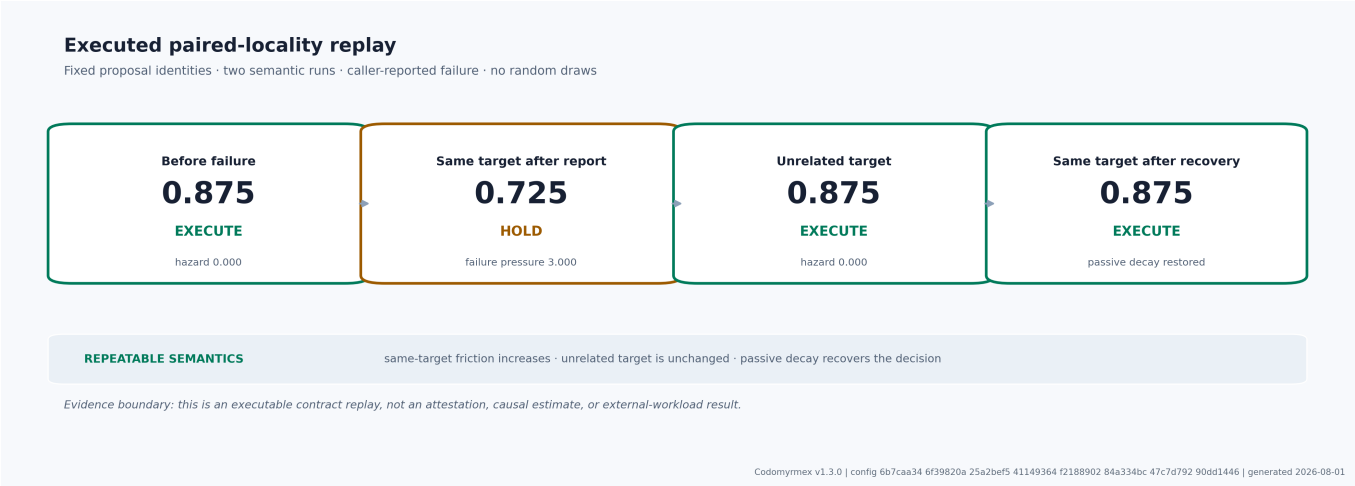

Figure 6: Fixed-input paired-locality replay derived from real Colony Kernel subsystems. The same-target score changes from 0.875/EXECUTE to 0.725/HOLD after a caller-reported failure, while the unrelated target remains at 0.875/EXECUTE with effective hazard 0.000, and passive decay restores 0.875/EXECUTE. The replay is repeated for semantic equality; it is not an attestation, causal estimate, or external workload result. Current default/illustrative policy values; configurable, not empirically calibrated.

The first stage shows an EXECUTE decision for a clear target. After a caller-reported failure, the same target moves to HOLD while a paired unrelated target stays at EXECUTE. Passive decay then restores the original decision. Stage order, decision words, score labels, and arrows carry the comparison; colours are redundant and the fixture is not causal or externally attested.

5.3 Gate landscape and attainable scores

The ordinary gate score has 4 non-negative weighted components. Because trust credit, hazard credit, and completeness are discrete, not every real number in $[0.0, 1.0]$ is attainable.

For a lower-tier authorized agent with clear budget and hazard, the score is

$$g(c) = w_b + w_\rho + w_u\, u + w_c c, \tag{21}$$

Here $b = 1$, $\rho(h) = 1$, and $u = 0.5$ for the stated clear-field, lower-tier case; the only varying term is the attainable completeness credit c.

Each coefficient and the lower trust credit are injected from the live gate. Runtime completeness has a finite attainable set because each missing field incurs the generated penalty. Thus the threshold is exact while the discrete input mapping can create gaps; the generated cases below expose those gaps without a second arithmetic source in prose.

Table 23 records representative, formula-checked cases.

Table 23: Representative hard-override and ordinary gate cases.

Condition	Budget credit	Hazard credit	Trust credit	Completeness	Score	Decision
SANDBOX, otherwise clear	—	—	—	—	0.000	REFUSE override
Trust 0.29, otherwise clear	—	—	—	—	0.000	REFUSE override
Lower trust, clear, no completeness fields	1.00	1.00	0.50	0.00	0.725	HOLD
Lower trust, clear, one of three fields present	1.00	1.00	0.50	0.30	0.770	EXECUTE
Lower trust, medium hazard, complete	1.00	0.50	0.50	1.00	0.725	HOLD
Full trust, high hazard, complete	1.00	0.00	1.00	1.00	0.700	HOLD
Full trust, clear, complete	1.00	1.00	1.00	1.00	1.000	EXECUTE

Figure 7 visualizes the controlled slice with budget and completeness fixed at 1.0 and no recent-failure penalty. The vertical discontinuities come from the trust hard floor and trust-credit tier; the horizontal discontinuities come from effective hazard thresholds at 3.0 and 6.0. The pressure axis should be read as $\max(\text{RISK}, \text{FAILURE})$, not RISK alone.

Trust increases from left to right and effective local hazard increases from bottom to top. A dark left band marks the hard trust-floor refusal region. Beyond it, labelled REFUSE, HOLD, and EXECUTE regions change at the configured hazard and score boundaries. Region position, printed names, and colour all encode the decision; the grid is formula-derived, not an observed distribution.

5.4 Trust accounting path

A new profile begins at trust 0.1 and role SANDBOX. Along the artificial all-success path, each recorded clean outcome adds 0.04. Table 24 reports exact checkpoints.

Table 24: Deterministic all-success trust path from the default profile.

Recorded outcomes	Trust	Inferred role	Gate implication
0	0.100	SANDBOX	Role override refuses
3	0.220	REPAIR_ANT	Role changes, but trust remains below the gate floor
5	0.300	REPAIR_ANT	Ordinary scoring reachable
6	0.340	REPAIR_ANT	Ordinary scoring reachable
9	0.460	MEMORY_ANT	Ordinary scoring reachable
12	0.580	DISPATCHER	Ordinary scoring reachable

Recorded outcomes	Trust	Inferred role	Gate implication
15	0.700	GUARD_ANT	Ordinary scoring reachable

The role ladder's first threshold (0.2) is below the gate's independent trust hard floor (0.3). Exiting SANDBOX therefore does not itself authorize a proposal.

Figure 8 plots this deterministic path. It is not a convergence plot: alternating or stochastic outcomes need not converge, and the current constant-step clipped update has no restoring term.

A monotone stair-step line starts at sandbox trust and rises after each caller-reported clean outcome. Horizontal threshold lines and directly named background bands show the inferred roles. A separate gate hard-floor line demonstrates that the first role promotion does not itself make ordinary scoring reachable. The path is a deterministic fixture, not a population trend.

The fixture is also not an attested credential protocol. The ordinary MCP outcome tool does not require a matching prior EXECUTE record, so an operator or client can submit the reports that drive this path. Local lifecycle attestation can protect linkage, but does not independently verify competence or external execution. This remains a deployment blocker for adversarial trust.

5.5 Subtractive signal dynamics

The field subtracts 0.3, 0.1, or 0.02 per tick for FAST, NORMAL, or SLOW traces, respectively, then removes traces at zero. Table 25 gives the exact integer-tick path for a unit deposit with no reinforcement.

Table 25: Exact passive decay of a unit trace under each class.

Tick	FAST	NORMAL	SLOW
0	1.00	1.00	1.00
1	0.70	0.90	0.98
2	0.40	0.80	0.96
4	0.00	0.60	0.92
8	0.00	0.20	0.84
10	0.00	0.00	0.80

Accordingly, a FAST unit trace has lost 100% after 10 ticks, while a SLOW unit trace retains 80%. Figure 1 plots the same recurrence continuously for legibility and states the integer-tick runtime interpretation in its caption.

5.6 MCP state and interface boundary

The package exposes the 8 MCP tools summarized in Table 26.

Table 26: Colony Kernel MCP interface and state effects.

Tool	State effect
`colony_propose_action`	Runs falsification and gate evaluation; may deposit traces
`colony_record_outcome`	Accepts a reported outcome; updates memory, budget, trust, role, and traces
`colony_agent_profile`	Reads one profile
`colony_status`	Reads the kernel snapshot
`colony_pheromone_query`	Reads typed pressure
`colony_falsify_plan`	Runs deterministic heuristic review; may deposit traces
`colony_pruning_report`	Reads pruning candidates
`colony_tick`	Advances decay and period-reset checks

Requests and responses are self-contained JSON-shaped documents, but calls are not stateless. A module-level `ColonyKernel` singleton shares state across calls within one server process. By default its consequence database is `:memory:` and its field is

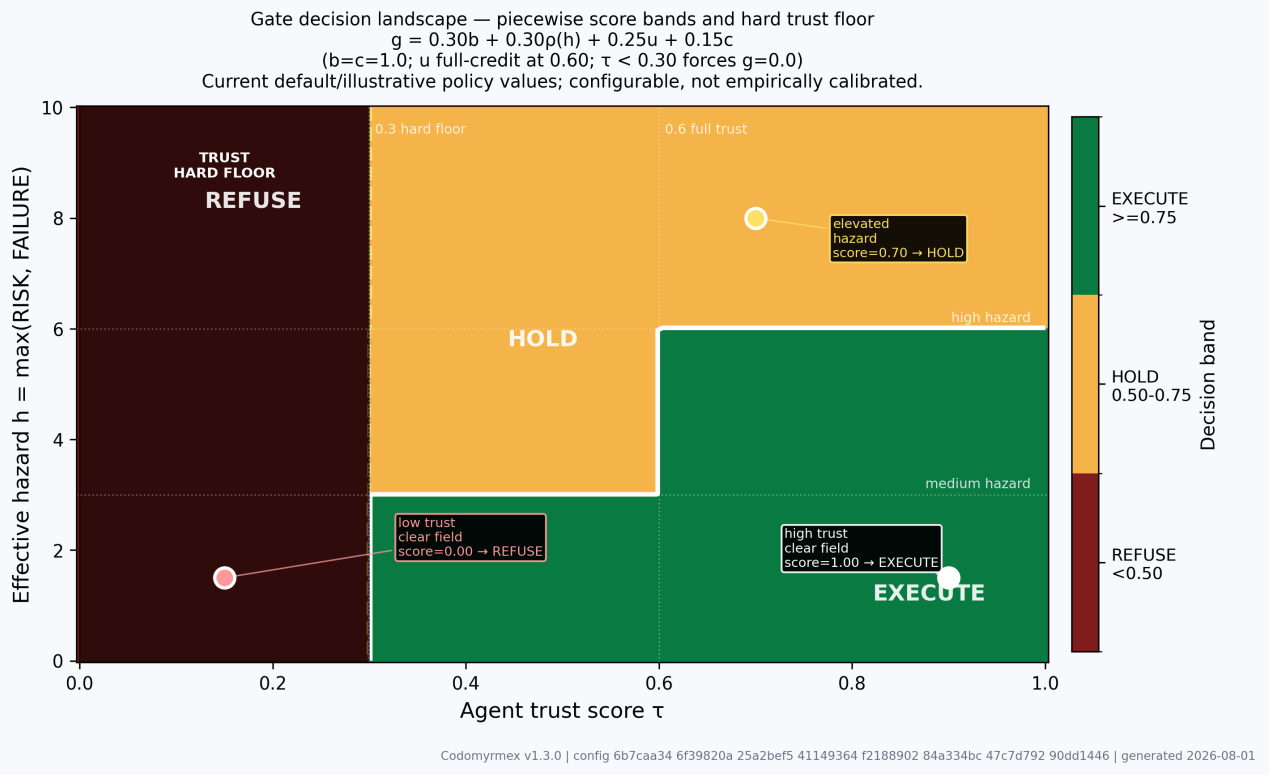

Figure 7: Formula-derived gate decision landscape over trust and effective local hazard pressure, where hazard is the maximum of RISK and FAILURE. Budget and completeness are fixed at 1.0; configured hazard boundaries are 3.0 and 6.0, and trust below 0.3 forces refusal. Colours encode configured decision bands redundantly with in-region decision labels and threshold annotations. This is an analytical policy map, not an observed proposal distribution. Current default/illustrative policy values; configurable, not empirically calibrated.

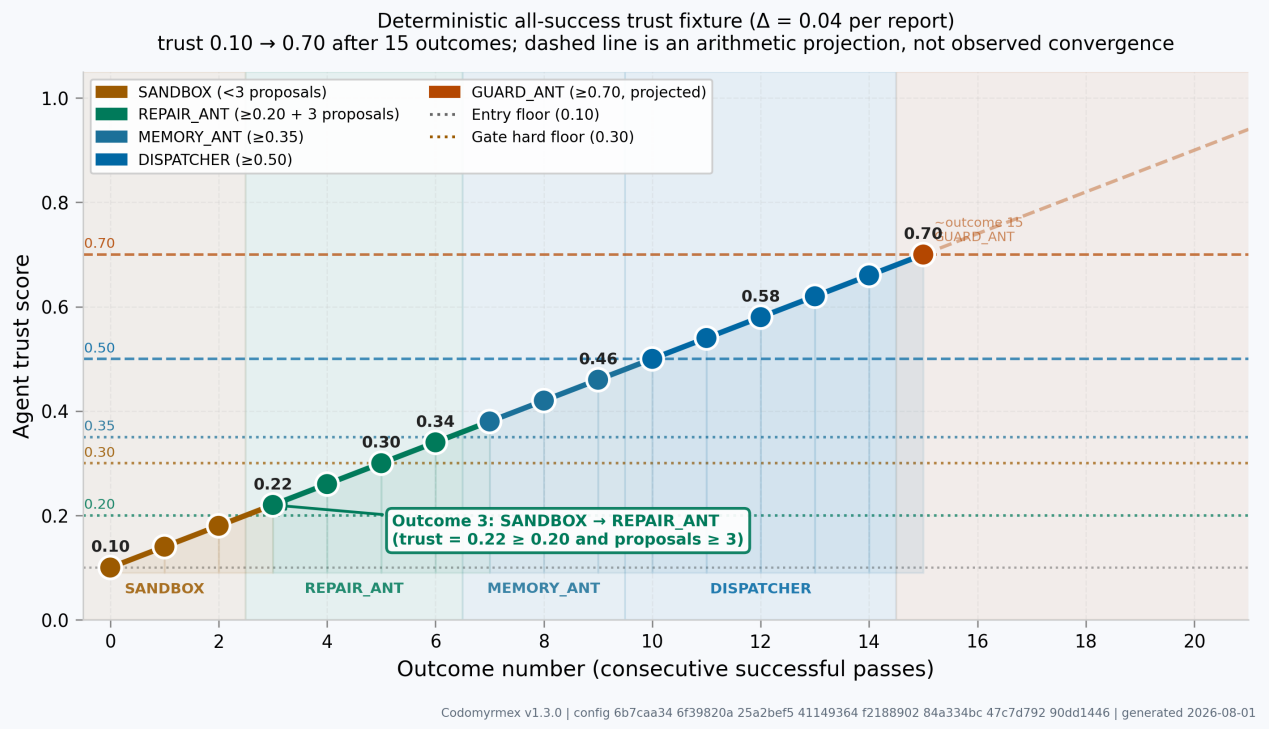

Figure 8: Deterministic trust trajectory under consecutive caller-reported clean outcomes through the configured 15-outcome horizon. Shaded bands show inferred role labels; the independent generated gate floor means the first role promotion does not yet make ordinary gate scoring reachable. Points are analytical applications of the fixed update, not population measurements or evidence of stochastic convergence. Current default/illustrative policy values; configurable, not empirically calibrated.

in-memory; restarting the process loses both. Supplying a file-backed SQLite path can persist consequence records, but the pheromone field still has no restart-persistent backend.

The default MCP compatibility path still has a proposal–outcome linkage gap: `colony_propose_action` does not return a durable authorization record consumed by `colony_record_outcome`, and the outcome tool synthesizes a proposal from caller input. The additive `AttestationLedger` provides proposal, verdict, authorization, receipt, outcome, rejection, and error events and rejects duplicate nonces. Optional mode records proposal/verdict events and offers a fully linked outcome route while preserving the ordinary caller-report method. Required mode rejects that ordinary method and makes the linked route mandatory. The default MCP singleton does not enable either mode, and even a valid local chain does not independently observe external actuation. This release therefore does not claim that every outcome report is authenticated or externally true.

5.7 What has not been measured

The current release has not executed the proposed 20-trial, 4-condition benchmark. Consequently:

- the configured trial count is a protocol parameter, not a sample size already observed;
- no refusal percentage is reported as an empirical result;
- no baseline comparison or hypothesis test has been run;
- no throughput, error-rate, budget-efficiency, or trust-stability effect size is claimed; and
- the figures in this paper are code-taxonomy, formula-derived, or deterministic-fixture visualizations—not plots of an external agent population.

This boundary is a result in its own right: it identifies exactly what the internal contract suite establishes and what the next empirical study must supply.

6 Scope, Related Work, and Claim Boundaries

The numeric values shown here are current implementation defaults or illustrative initial settings, not universal constants, fitted parameters, or empirical estimates. They are configurable and can be tuned or replaced through the corresponding runtime policy, configuration, or presentation inputs; after a change, regenerate the tests, tables, figures, and manuscript before interpreting the result. This distinction is central to the scholarship: the paper contributes an inspectable implementation and a falsifiable evaluation agenda, not a claim that a hand-specified policy has already been calibrated or shown superior.

6.1 Unit of Analysis

The unit of analysis in this paper is the Colony Kernel, not every subsystem distributed with Codomyrmex. The broader repository includes agent, orchestration, model-integration, identity, and other facilities. The claims made here concern the kernel path that combines proposal data, budget state, pheromone readings, consequence-derived trust, role inference, falsification findings, and a ternary gate decision.

This distinction matters for both novelty and evaluation. The Colony Kernel can be used as a control-plane component, but the present artifact does not show that every Codomyrmex action is mediated by it, that external clients cannot bypass it, or that a gate decision is enforced by an operating-system or cloud authorization layer.

6.2 Agentic Software Engineering

SWE-bench evaluates model-generated patches against real repository issues, while SWE-agent shows that the agent-computer interface materially shapes software-engineering performance (Yang et al. 2024b, 2024a). These projects motivate evaluating a control plane in the same environments where agents inspect files, run tools, and modify persistent state. They do not provide evidence for the Colony Kernel until the kernel is actually evaluated as part of such a workflow.

LangGraph, AutoGen, and CrewAI provide stateful graphs, conversational multi-agent composition, and role-oriented orchestration, respectively (AI 2024; Wu et al. 2023; Inc. 2024). Their current designs include state or memory facilities, so the Colony Kernel's positioning does not depend on claiming that other frameworks are stateless or lack cross-session mechanisms. The narrower distinction is architectural: the kernel exposes a deterministic proposal-evaluation path over explicit budget, effective local hazard, trust, and completeness inputs. Whether that path improves outcomes when integrated with another runtime is an empirical question, not a property established by juxtaposing feature lists.

The Model Context Protocol supplies a standard tool-exposure interface. The referenced official line is revision 2026-07-28 (Model Context Protocol Contributors 2026), whereas the current Codomyrmex client and server advertise `2025-06-18`. The revisions

match: `false`; therefore not established: Codomyrmex advertises 2025-06-18, while the referenced official revision is 2026-07-28. This is a source-level revision measurement, not a conformance suite. The kernel tools are callable through Codomyrmex's implemented interface, but compatibility with the current official line must be tested rather than assumed. Protocol exposure is also not enforcement: a client or tool path that does not invoke the gate is outside the protection claimed here.

6.3 What the Colony Kernel Is—and Is Not

A deterministic governance prototype. The gate converts a specified kernel state and proposal into EXECUTE, HOLD, or REFUSE. This makes the decision path inspectable and testable.

Not a complete agent runtime. The kernel itself does not generate proposals, choose models, execute arbitrary tools, or guarantee that a downstream runtime obeys its verdict. Other Codomyrmex packages may provide adjacent capabilities, but they are not evidence for the kernel claims in this paper.

Process-lifetime by default. The MCP module owns a single kernel instance within one process. Consequence memory defaults to SQLite in-memory mode, and the pheromone field is in memory. A caller can choose a file-backed database for consequence records; the default does not provide cross-session persistence, and file-backed consequence records do not by themselves persist the whole kernel or pheromone field.

Not a per-role authorization system. RoleAdapter infers labels from trust and proposal count, and SANDBOX is a hard gate condition. The other role labels do not currently define or enforce an action-by-role permission matrix.

Not a security boundary. Trust scores are mutable state associated with caller-supplied agent identifiers, not unforgeable capability tokens. The default outcome path remains caller-reported. Optional `AttestationLedger` mode offers a fully linked local route while preserving the ordinary method; required mode rejects the ordinary method and requires proposal, EXECUTE verdict, authorization, execution receipt, and outcome events. The ledger does not prove that an external observer saw a safe or useful action and is not automatically enabled for every interface.

Not production- or scale-validated. The checked-in tests exercise internal contracts. The manuscript does not release the repeated-trial traces, production replays, concurrent load results, or external benchmark runs needed for effectiveness or scaling claims.

Not an integrated Active Inference implementation. The core kernel remains deterministic and has no posterior or expected-free-energy policy optimizer. A separate, explicit probabilistic adapter now declares states, observations, likelihoods, priors, transitions, preferences, horizon, and seed for offline research; its presence does not turn gate scores into probabilities or connect that model to production actuation. The comparison in Section 7 remains a design crosswalk.

6.4 Stigmergy and Environmental Traces

Grassé introduced stigmergy to describe coordination mediated by changes to a shared environment (Grassé 1959). Digital-pheromone systems later adapted this idea to software-agent coordination (Parunak 1997), while ant-colony optimization formalized reinforcement and evaporation for search (Dorigo and Stützle 2004). The Colony Kernel borrows the environmental-trace idea; it does not implement an ant-colony optimization algorithm.

The TraceField stores strengths under compound location and signal-type keys. Evaporation occurs only when the kernel is explicitly ticked, subtracting a configured amount and deleting depleted markers. This is a deterministic, discrete-time store rather than a continuous diffusion field. Its coordination scope is also bounded by process lifetime in the default MCP configuration.

Signal types must not be conflated. A failed outcome deposits FAILURE and changes the responsible agent's consequence history; prospective checks may deposit RISK. The gate retains these raw readings for diagnosis and uses their maximum as effective local hazard. Thus a reported failure can tighten a later same-target decision without being silently relabeled as RISK. This coupling is process-local and only as trustworthy as the unattested report that created the FAILURE signal.

6.5 Computational Trust and Role Labels

Computational-trust research treats trust as contextual, evidence-dependent, and distinct from simple identity (Marsh 1994; Sabater and Sierra 2005). FIRE combines direct interaction with witness, role-based, and certified evidence (Huynh et al. 2006). These sources provide comparison points, not validation of the Colony Kernel's scalar score.

The current kernel uses fixed outcome deltas and clamps the resulting score to a bounded range. It does not estimate a calibrated probability of competence, benevolence, integrity, or future safety. It also does not import witness reputation or certified

credentials into the trust update. Two agents with the same score but different sample sizes can therefore receive the same trust tier, and an outcome record has meaning only to the extent that the caller and record are trustworthy.

Role inference is deterministic categorization over this heuristic history. The labels may be useful for routing, explanation, or future policy definition, but only implemented gate conditions have authorization effect. Calling the labels a permission ladder would overstate the artifact until each action class is checked against an explicit policy.

6.6 Security Boundary

Capability security requires authority that cannot be obtained merely by naming another principal, while least privilege requires each component to receive only the authority it needs (Miller and Shapiro 2003; Saltzer and Schroeder 1975). A scalar trust score indexed by agent identifier is not such a capability. The gate may contribute evidence to a broader authorization decision, but it must be paired with authenticated identities, non-bypassable mediation, sandboxing, constrained credentials, and downstream policy enforcement.

NIST Zero Trust Architecture likewise requires a broader resource-centric control system, including policy decision and enforcement points, identity and device evidence, and continuous evaluation (Rose et al. 2020). The Colony Kernel is compatible with the general practice of reevaluating proposed actions, but it is not an implementation or certification of NIST SP 800-207.

6.6.1 Threat-informed evaluation

Tool-using agents can be redirected by untrusted content, can misuse available tools, and can fail in high-impact simulated environments. AgentDojo and InjecAgent operationalize prompt-injection attacks against tool-integrated agents, and ToolEmu evaluates risks in high-stakes tool use (Debenedetti et al. 2024; Zhan et al. 2024; Ruan et al. 2023). These are appropriate future falsification workloads. They have not been run in this manuscript, so they should not be cited as evidence that the gate reduces attack success.

Runtime-assurance work provides another useful comparison. Simplex architectures interpose a decision mechanism capable of switching from an advanced controller to a verified-safe baseline (Seto et al. 1998). The Colony Kernel also interposes a decision, but it has no verified fallback controller or formal safety invariant. The comparison identifies an engineering direction rather than an inherited guarantee.

Table 27: External validation agenda; every row is future work rather than a reported benchmark.

Evaluation question	Suitable evidence	Status in this paper
Does the gate improve repository-task outcomes?	Controlled SWE-bench or SWE-agent integration with a baseline	Not run
Does it reduce unsafe tool use under indirect injection?	AgentDojo or InjecAgent attack-success comparison	Not run
Does it reduce high-impact tool failures?	ToolEmu scenarios with linked proposal, verdict, and outcome traces	Not run
Can the local lifecycle chain reject forgery and replay?	Signed/hash-linked deterministic ledger tests	Supported for local ledger mechanics
Can clients bypass mediation or forge external actuation evidence?	Authenticated end-to-end deployment adversarial tests	Not run
Does state survive restart and concurrent access?	Persistence and concurrency tests under a declared deployment configuration	Not established for the default MCP service
Are HOLD and REFUSE calibrated to downstream harm?	Held-out outcomes with calibration and utility analysis	Not evaluated

Table 27 states the minimum comparisons needed before making effectiveness or deployment claims.

6.7 Active Inference and Free Energy

Friston's Free Energy Principle (Friston 2010) frames intelligent behavior as minimization of variational free energy—the gap between an agent's generative model and its sensory observations. A full mapping of the Colony Kernel onto this framework would require an explicit generative model over observations and hidden states, an implemented posterior approximation, and policy selection under expected free energy (Friston et al. 2017).

Section 7 develops this analogy explicitly by proposing a generative-model interpretation of colony observations, hidden states, and policy selection. That treatment is a conceptual reconstruction of the implemented heuristics, not evidence that the current kernel performs variational Bayesian inference or minimizes canonical expected free energy. High failure pheromone concentrations can be read as coarse, persistent prediction-error signals, but the mapping remains an engineering interpretation rather than a proof of formal equivalence.

6.8 Explicit Limitations and Future Work

The benchmark protocol in Section 4 remains unexecuted as a repeated comparative study. Configured agent counts, scenarios, and expected rates are protocol parameters or analytical fixtures, not observations. A publication-quality experiment requires released traces, explicit baselines, linked proposal-to-outcome records, predeclared outcomes, and an analysis appropriate to ternary gate decisions.

The current system is synchronous and single-process by default. It has no demonstrated throughput envelope, distributed consistency model, or merge protocol for independent pheromone fields and trust histories. File-backed consequence storage can be evaluated as one deployment option, but it should not be described as persistence of the complete control plane.

The gate weights, trust deltas, role thresholds, decay amounts, and falsification checks are hand-authored heuristics. They are auditable, but auditability is not calibration. Future learning or optimization should predict downstream repair, harm, or policy violation on held-out data; fitting a model to reproduce the existing EXECUTE/HOLD/REFUSE labels would merely imitate the current rule.

Adjacent wallet, HMAC challenge-response, physiological-signal, persona, and spatial world-model modules are outside the evaluated gate path. This paper makes no zero-knowledge, wallet-ownership, coercion-detection, physiological-authentication, or trajectory-aware gating claim on their behalf.

The resulting position is deliberately modest. The Colony Kernel is an inspectable software experiment in consequence-aware gating. Its deterministic contracts are testable now. Persistence across sessions, enforceable role authority, resistance to adversarial clients, and improved safety on realistic workloads remain implementation and evaluation targets. The relevant comparison with prior work is therefore at the level of control-plane decomposition and measurement design; it should not be read as an empirical comparison with the cited systems.

7 Active Inference: Bounded Crosswalk and Upgrade Path

Active inference provides a vocabulary for relating perception, belief, action, and learning under uncertainty (Friston 2010; Friston et al. 2017). That vocabulary can help ask sharper questions about the Colony Kernel. It does not, by itself, make the kernel an active-inference system.

The numeric values shown here are current implementation defaults or illustrative initial settings, not universal constants, fitted parameters, or empirical estimates. They are configurable and can be tuned or replaced through the corresponding runtime policy, configuration, or presentation inputs; after a change, regenerate the tests, tables, figures, and manuscript before interpreting the result. The active-inference correspondence below is therefore a research scaffold: it identifies missing model components and testable upgrade steps, not a retrospective theoretical certification of the current heuristic gate. The deterministic settings used in this crosswalk are example/initial engineering values that can be tuned through their owning code or configuration; tuning them does not create a probabilistic model or an empirical Active Inference result.

The core implementation is deterministic and heuristic. It does not integrate a probabilistic generative model, maintain a variational posterior, estimate precision from data, evaluate policies over a planning horizon, or minimize expected free energy. A separate offline research adapter declares these objects for explicit experiments, but it is not called by the production gate and does not relabel deterministic scores. This section therefore presents a conceptual crosswalk and an implementation agenda. Its correspondences are analogies to be tested, not formal equivalences or empirical results.

7.1 Canonical Requirements

In a canonical variational treatment, an agent specifies a joint density $p(o, s)$ over observations o and latent states s, and maintains an approximate posterior $q(s)$. Variational free energy may be written as

$$\mathcal{F}[q; o] = D_{\mathrm{KL}}(q(s) \,\|\, p(s \mid o)) - \log p(o), \tag{22}$$

so minimizing $\mathcal{F}$ in Equation 22 with respect to q makes the approximation approach the posterior under the stated model. Active inference additionally evaluates policies using predicted future observations, preferences, and expected free energy (Friston 2010; Friston et al. 2017).

For a Colony Kernel implementation to satisfy that description, the software would need to represent at least:

1. a declared observation model and latent-state space;
2. priors and an updateable posterior approximation;
3. policies containing possible future action sequences;
4. preferences or costs over predicted outcomes; and
5. an inference procedure that computes or approximates posterior beliefs and policy scores.

The checked-in kernel has deterministic state variables and thresholds instead. Giving those variables names borrowed from active inference does not supply the missing model or inference procedure.

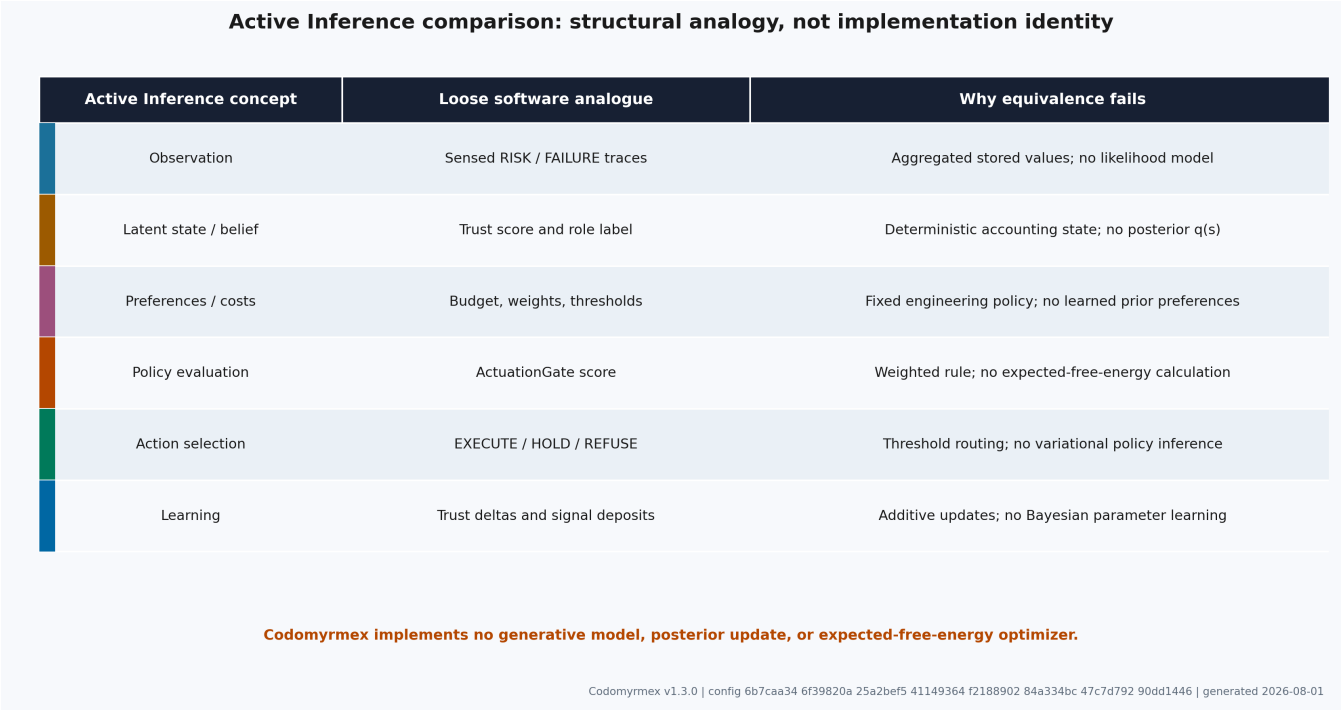

Figure 9: Schematic vocabulary crosswalk between Free Energy Principle terms and Colony Kernel artifacts. Each row names an FEP concept, the kernel artifact proposed as an analogue, and the intended engineering intuition. The correspondence is conceptual: the current kernel does not implement Bayesian inference or formal equivalence with expected free energy.

Each horizontal row begins with an FEP concept, points to a specific kernel artifact, and ends with an engineering interpretation. The repeated three-part layout and text labels carry the mapping; coloured row accents only aid scanning. A warning beneath the rows states that the current deterministic kernel is not a Bayesian controller and the correspondences are analogies, not equivalences.

The crosswalk in Figure 9 is therefore a reading aid, not a model diagram. The crosswalk identifies where a future probabilistic implementation might attach to the existing interfaces.

7.2 Status of the Proposed Correspondences

Table 28: Conceptual correspondences and implementation status; the final column is the claim boundary.

Active Inference term	Suggested Colony Kernel analogue	Status in the current implementation
Observation o	Strengths returned by TraceField sensing	Concrete numeric state, but not samples from a declared likelihood
Hidden state s	Agent competence or location risk	Interpretive latent quantities; no corresponding random variables are represented

Active Inference term	Suggested Colony Kernel analogue	Status in the current implementation
Approximate posterior $q(s)$	Agent trust score or inferred role	Fixed-delta summary and deterministic label, not a probability distribution
Policy π	ActionProposal	One proposed action, not a policy sequence evaluated over future states
Expected free energy $G(\pi)$	Gate score	Hand-weighted threshold score, not an EFE calculation
Precision weighting	Signal source multipliers	Configured deposit weights, not estimated inverse variances
Learning	Consequence-derived trust update	Clipped heuristic increments, not posterior or parameter learning
Markov blanket	ActuationGate boundary	Software mediation metaphor, not a demonstrated conditional-independence blanket

Table 28 makes each analogy's non-equivalence explicit.

7.2.1 Trust is not a posterior

The trust score summarizes selected outcome fields by adding fixed positive and negative deltas and clamping the result. It does not retain the sufficient statistics of a declared likelihood, and two agents with different evidence volumes can share the same score. A Beta distribution can be proposed as a future model for binary outcomes, but that model would require explicit assumptions about exchangeability, dependence, nonstationarity, repair events, and human feedback. It cannot be inferred from the current scalar update.

The RoleAdapter adds no Bayesian step. It maps the heuristic score and proposal count to a named tier. Those names remain operating labels; apart from implemented gate conditions such as SANDBOX, they do not constitute a probabilistic belief state or a per-action authorization policy.

7.2.2 Signal strength is not prediction error or precision

The pheromone field is useful as a stigmergic store: one operation changes shared environmental state that a later operation can sense. This follows the broad environmental coordination idea in stigmergy (Grassé 1959; Parunak 1997). Within the default MCP server, however, that shared state lasts only for the process and does not constitute a shared generative model.

The signal semantics are also discrete implementation choices. FAILURE and RISK occupy different keys, while the gate computes effective hazard as their maximum. A reported FAILURE can therefore lower the next same-target gate score without becoming a Bayesian risk prior or changing the stored RISK channel. Likewise, source multipliers express configured deposit strength; without a noise model or estimated variance, they are not statistical precision.

Evaporation subtracts a configured amount on each explicit tick and removes depleted markers. It is not exponential Bayesian forgetting, and elapsed wall-clock time has no effect unless a caller advances the kernel.

7.3 Gate Decisions and Epistemic Action

The gate combines budget approval, tiered effective hazard, trust, and completeness terms, plus hard conditions, into a deterministic score. No probability distribution over future outcomes is generated, and no alternative policy sequence is rolled out. The gate score should therefore not be identified with $1 - G(\pi)$, with a probability of safety, or with an upper bound on harm.

HOLD admits a limited epistemic analogy. A caller may respond to HOLD by supplying better evidence or revising a plan, which can reduce uncertainty for a human or downstream system. The kernel itself does not choose an information-gathering action, compute expected information gain, or guarantee that the revision is informative. HOLD is a request for revision under deterministic rules, not a literal active-inference policy.

EXECUTE has the same boundary. It means that the proposal cleared the current rule under the current state. It does not mean that expected free energy was minimized or that the action is safe.

7.4 Environmental Embedding and Markov Blankets

Environmental traces can coordinate successive operations without direct messages, and variational-ecology work offers one theoretical language for studying coupled agents and environments (Ramstead et al. 2019). The Colony Kernel provides a small engineering example of environmental state being sensed after earlier deposits. Demonstrating collective inference would require more: agent-specific models, a shared observation process, and evidence that updates improve posterior prediction or policy selection.

A Markov blanket is a conditional-independence structure in a probabilistic graphical model (Pearl 1988). The ActuationGate is a software boundary between a proposal and a verdict, but this alone does not establish the conditional independencies of a Markov blanket. Calling the gate a blanket is acceptable only as a clearly marked metaphor for mediation.

The process boundary matters here as well. The default MCP singleton lets multiple calls within one process encounter the same in-memory field. Restarting the process removes that field, and a file-backed consequence database does not restore it. Cross-session collective belief propagation is therefore not an implemented property.

7.5 From Analogy to an Active Inference Model

Turning the analogy into a scientific model would require a new implementation and a new evaluation, not a relabeling of the existing score.

Specify the generative model. Define observable events, latent quantities, likelihoods, priors, conditional independencies, and the time model. FAILURE, RISK, trust, and repair must have distinct semantics in that model.

Link evidence to actions. Each observation used for inference should be bound to an approved proposal, executed action, actor identity, and measured outcome. The current ability to submit an outcome without such a chain is unsuitable as an inference dataset.

Implement posterior inference. Maintain an explicit posterior approximation and test it using simulation-based calibration, posterior predictive checks, and held-out prediction. A scalar trust score may remain an interface projection, but it should not be treated as the posterior itself.

Define policies and preferences. Represent alternative action sequences, future state transitions, and preferred outcomes. Then compute or approximate expected free energy and compare its choices with both the deterministic gate and observed downstream outcomes.

Calibrate against consequences. Learning should target repair cost, policy violation, or another independently observed outcome. Training a model to predict the existing EXECUTE/HOLD/REFUSE label would reproduce the current rule rather than validate it.

Run a comparative experiment. Release seeds, traces, baselines, and failure analyses for the deterministic gate and the probabilistic alternative. Evidence that the latter improves calibration or utility would support the upgrade; conceptual similarity alone would not.

These steps could yield an active-inference-inspired successor. They would still not prove that the software is formally equivalent to a biological active-inference model; that is a separate mathematical claim requiring explicit definitions and proof.

7.6 Summary

Active inference is useful here as a question generator. It asks what the kernel observes, which uncertainty it represents, how evidence changes belief, and how present action is valued against possible futures. The current answers are deterministic strengths, fixed trust deltas, categorical role labels, and thresholded gate decisions.

That architecture may be a practical substrate for later probabilistic work, but the present Colony Kernel performs neither variational Bayesian inference nor expected-free- energy minimization. The honest correspondence is schematic: a map of where formal components might be built, with the unmapped territory left visible.

8 Reproducibility Chain, Provenance, and Limits

This section describes the evidence that the checked-in manuscript route actually regenerates. It is a scoped build record, not a claim that every Codomyrmex module was tested, that the Git worktree had no uncommitted edits, or that the proposed external benchmark in Section 4.1 was executed.

The numeric values shown here are current implementation defaults or illustrative initial settings, not universal constants, fitted parameters, or empirical estimates. They are configurable and can be tuned or replaced through the corresponding runtime policy, configuration, or presentation inputs; after a change, regenerate the tests, tables, figures, and manuscript before interpreting the result. The generated snapshot binds the rendered numbers to one configuration and source state; it does not turn those values into fitted or externally validated parameters.

8.1 Evidence boundary

Table 29 separates the claims supported by the release route from claims that require additional evidence.

Table 29: Scope of the reproducibility evidence.

Evidence surface	Regenerated by this route	Not established
Colony Kernel tests	Pass/fail count for `tests/unit/colony_kernel/`	Correctness outside exercised cases
Branch coverage	Coverage of `src/codomyrmex/colony_kernel` against the configured 60.0% floor	Coverage of the full Codomyrmex package
Static checks	Ruff and ty status for `src/codomyrmex/colony_kernel`	Repository-wide lint or type cleanliness
Manuscript variables	Token map computed from current files, configuration, and gate outputs	A signed or independently attested release
Figures	18 regenerated visual assets sourced from the variable snapshot and documented constants	Measurements from an external agent population
Figure accessibility	Configured captions, concise alternatives, extended descriptions, redundant encodings, and palette-contrast checks	Usability with every assistive technology, display, print process, or reader
Claim audit	Machine-readable claim classes, evidence paths, citations, and boundaries	A claim ledger does not create evidence missing from the cited paths
Bibliography audit	Pandoc citations separated from cross-references; cited primary locators resolved and unused records rejected	Registry resolution does not validate every argument made by a source
Paired replay	Fixed-input semantic replay with repeat-run equality and JSON digest	External-actuation attestation, concurrency, restart durability, or external effectiveness
Local attestation	Signed, hash-linked proposal/verdict/authorization/receipt/outcome fixture	Independent observation of external actuation or deployment safety
Render	Hydrated Markdown, semantic HTML, content PDF, and bookended distribution PDF	Byte-identical output across machines and dates
PDF conformance	`qpdf --check` plus independent veraPDF PDF/UA validation receipts for the current content and distribution PDFs	Universal usability across assistive technologies, displays, print processes, or readers

The current content and distribution PDFs have an independent veraPDF PDF/UA pass recorded in `output/validation/paper-content-pdf-validation.json` and `output/validation/paper-pdf-validation.json`. This is artifact-specific conformance evidence for the rendered files; it does not establish universal usability across assistive technologies, displays, print processes, or readers.

The variable generator fails when the scoped pytest process, branch-coverage threshold, Ruff, or ty fails. Rendering normally invokes this generator before consuming the hydrated manuscript. “Fail closed” in this paper therefore refers to those named, scoped gates. It does not imply that unrelated repository tests or external benchmark requirements are part of the same decision.

8.2 Configuration identity

Table 30 records the manuscript configuration identity injected into this render.

Table 30: Configuration identity embedded in the rendered manuscript.

Property	Rendered value	Source
Configuration digest	`6b7caa34 6f39820a 25a2bef5 41149364 f2188902 84a334bc 47c7d792 90dd1446`	Full SHA-256 over the raw bytes of `docs/manuscript/config.yaml`
Experiment replay seed	`0`	Explicit protocol input; the current paired replay uses no random draws
Manuscript version	1.3.0	`paper.version` in the same YAML file
GitHub release tag	`v1.3.0-paper`	`publication.release_tag` in the same YAML file
First author	Daniel Ari Friedman	First entry under `authors`
Keywords	ai-agents, model-context-protocol, mcp, multi-agent, orchestration, colony-control-plane, stigmergy, artificial-ecology, agentic-software-engineering, falsification-worker, actuation-gate, trust-scoring	`keywords` list
Generation time	2026-08-01T23:11:17Z	UTC time when the variable map was computed

Protocol compatibility is measured rather than inferred from the MCP name. The referenced official line is revision 2026-07-28 (Model Context Protocol Contributors 2026), while the current client and server sources advertise `2025-06-18`. Equality is `false`; the resulting status is not established: Codomyrmex advertises 2025-06-18, while the referenced official revision is 2026-07-28. This string-level check is provenance evidence, not a protocol conformance or interoperability test.

The configuration digest is a compact change detector for one file. The generated snapshot also records the source commit, worktree state, environment fingerprint, the first-party Colony Kernel/manuscript source digest, lockfile/project hashes, and the authoritative inventory digest below. These values are integrity metadata, not signatures or authenticity proofs.

The generator reads the current filesystem. It can include uncommitted edits, and it does not reject such a state. A release that needs later reconstruction should record the commit and submodule revisions, `git status --porcelain=v1`, any retained diff, the full SHA-256 of `uv.lock`, and hashes of the emitted evidence and publication files. Table 32 records the identity fields retained for that reconstruction.

8.3 Generated evidence and artifact conventions

Table 31 reports live inventory values available to the manuscript. These are counts of files or definitions visible during generation; they are not all newly created artifacts.

Table 31: Live release inventory exposed to manuscript tokens.

Inventory	Rendered count	Counting rule
Colony Kernel test modules	20	`test_*.py` directly under `tests/unit/colony_kernel/`
Colony Kernel YAML files	3	`*.yaml` and `*.yml` directly under `config/colony_kernel/`
Colony Kernel MCP tools	8	`@mcp_tool` definitions in `mcp_tools.py`
Colony Kernel documentation files	3	`*.md` directly under `docs/modules/colony_kernel/`
Top-level Colony Kernel Python files	16	`*.py` directly under `src/codomyrmex/colony_kernel/`

Table 32: Reproducibility identity recorded for the generated evidence snapshot.

Reproducibility fact	Rendered value
Source commit	`9207ac24 d69f0d29 2a2c26cc baf823a2 2fb75d70`
Worktree dirty	`true`
Environment fingerprint	`dc82a6bf 4fb221c8 c277cf46 41a09910 6c8d86e3 89a5e6c5 77bea24e a79d419f`
Colony Kernel/manuscript source SHA-256	`1850ada7 bc641d88 e042427e 0f4a3cc7 804ccebc b0893049 f04996cb 2b5648b1`
`pyproject.toml` SHA-256	`f1d88718 3511e920 f268da51 45766ccc 6ff27308 7feadc89 fd0fb4c9 b256ff4e`
`uv.lock` SHA-256	`2b94343c 276c5322 c5e11b17 6d44f519 7c959bdb 173cb6c4 59cc2a00 a6ebb6c3`
Authoritative inventory SHA-256	`cbcd02e7 f50e985f d8d54dff 5aadbe1d ef90f3c3 82d7355f b8c02ac2 28e4cb40`
Inventory module / MCP file / decorator / workflow counts	`130` / `150` / `623` / `37`
Fixed-input replay artifact	`output/data/colony_kernel_replay.json`
Replay semantic digest	`af96389d d276627c 853eeae9 15a54d9a 536e13ff c4d7b74e 9e9f6277 d8b0aabc`
Replay record SHA-256	`b49f284d 3e311bcf ccb8f9b1 ce4f6639 719d34d1 f135ab0e 1e476190 d4dfdf26`
Replay file SHA-256	`d0799e32 5bc2f89a 0d9d8b09 81ddfd8f 9afd2b67 241f922b 7c10c279 7e8bb17b`
Replay assertions / decisions	repeatable=`true`; same-target=`HOLD`; unrelated=`EXECUTE`; recovered=`EXECUTE`

Long hexadecimal identifiers in this table are grouped with spaces only to create line-break opportunities in the rendered document. Remove whitespace before comparing them with the machine-readable replay, configuration, or provenance artifacts.

The project uses the following output conventions:

- `output/data/manuscript_variables.json` stores the complete rendered token map;
- `output/data/colony_kernel_coverage.json` stores the fresh scoped coverage report;
- `output/data/colony_kernel_replay.json` stores the fixed-input paired replay, semantic assertions, provenance, and digests;
- `output/data/bibliography_audit.json` stores cited/unused/missing inventories, primary locators, registry title checks, and any access-limited resolution;
- `output/manuscript/` stores token-resolved section copies plus `config.yaml` and the bibliography used by the renderer;
- `output/figures/` stores the 18 generated PNG figures and `figure_registry.json`, whose entries record caption, concise alternative, extended description, evidence class, byte size, and full SHA-256 for each PNG, while its schema-level provenance records the source commit, dirty state, and Colony Kernel / manuscript source digest;
- `docs/manuscript/claim_ledger.yaml` records the active claim audit. The integrity validator checks that every listed source/evidence path exists and that cited bibliography keys resolve;
- compilation writes `output/paper.html`, the unbookended `output/paper-content.pdf`, the bookended `output/pdf/codomyrmex_`
and PDF validation receipts; and
- `python -m codomyrmex.release publication prepare` writes the portable `output/release/codomyrmex-1.3.0/` bundle with citation metadata, Zenodo deposit metadata, detached checksums, reproducibility inputs, and `publication_manifest.j`

PDF presentation layout is configuration-backed. The current initial margin is `0.58in`; it is a tunable presentation setting, not a scientific constant or calibrated result. Changing it requires regenerating the variables, figures, HTML/PDF, and integrity manifest, followed by visual inspection. Layout changes do not alter the underlying runtime measurements.

The generated replay JSON is the machine-readable semantic evidence record for the paired fixture; its file digest is injected into the manuscript at generation time.

These paths are generated workspace outputs. Their presence alone does not show that they are current; the successful command log and artifact hashes are needed to bind them to a particular run.

8.4 Scoped quality gates

Table 33 repeats the same generated values reported in Table 21.

Table 33: Scoped quality-gate snapshot regenerated for the manuscript.

Gate	Rendered result	Exact scope
pytest	819 passed	`tests/unit/colony_kernel/`
branch coverage	76.6%	`src/codomyrmex/colony_kernel`; 60.0% project floor
Ruff	0 findings	`src/codomyrmex/colony_kernel`
ty	0 diagnostics	`src/codomyrmex/colony_kernel`

The generator deletes the prior coverage JSON before invoking pytest, requires a newly written report with a branch-coverage percentage, and raises on a non-zero subprocess result. Ruff and ty are also rerun rather than read from a cached scorecard. The test count is parsed from the same scoped pytest process, with a collection-only fallback if the summary cannot be parsed.

The suite introduces no prohibited mock framework and includes real value objects, filesystem cases, and both in-memory and SQLite-backed cases. That testing style checks more integration behavior than isolated substitutes would, but it neither reproduces a production deployment nor validates the truth of caller-reported outcomes.

8.5 Exact reproduction commands

The supported route begins at the Codomyrmex repository root:

```
uv sync --locked --group docs
uv run --locked python scripts/z_generate_manuscript_variables.py
uv run --locked python scripts/generate_manuscript_figures.py
uv run --locked --group docs python scripts/compile_manuscript.py \
  --manuscript-dir output/manuscript --output-dir output \
  --check --skip-generate
uv run --locked --group docs python scripts/compile_manuscript.py \
  --manuscript-dir output/manuscript --output-dir output \
  --pdf --bookends --pdf-engine lualatex --pdf-standard ua-2 --skip-generate
uv run --locked python scripts/validate_manuscript_integrity.py \
  --require-rendered --online-bibliography
```

The scoped evidence gates and package-wide static gates can be inspected directly:

```
uv run --locked pytest tests/unit/colony_kernel/ \
  --cov=src/codomyrmex/colony_kernel \
  --cov-branch \
  --cov-report=json:output/data/colony_kernel_coverage.json \
  --cov-report=term
uv run --locked ruff check src scripts tests
uv run --locked ruff format --check src scripts tests
uv run --locked ty check
uv run --locked python scripts/replay_colony_kernel.py
```

The variable producer isolates this scoped branch-coverage run from any parent pytest-cov session by assigning it a dedicated raw-data file and removing inherited pytest-cov bootstrap variables. This prevents a repository-wide statement-coverage run from attempting to combine incompatible branch-coverage data.

After those files exist, the detached publication bundle and non-mutating remote plans are produced and verified with:

```
uv run --locked python -m codomyrmex.release publication prepare \
  --validation-receipt output/data/bibliography_audit.json \
  --validation-receipt output/validation/paper-content-pdf-validation.json \
  --validation-receipt output/validation/paper-pdf-validation.json
uv run --locked python -m codomyrmex.release publication verify \
  output/release/codomyrmex-1.3.0
uv run --locked python scripts/validate_manuscript_integrity.py \
  --require-rendered --require-source-current
uv run --locked python -m codomyrmex.release publication plan \
  output/release/codomyrmex-1.3.0 --target github
```

```
uv run --locked python -m codomyrmex.release publication plan \
  output/release/codomyrmex-1.3.0 --target zenodo-sandbox
```

`scripts/compile_manuscript.py --check --skip-generate` checks the hydrated section set for unresolved double-brace variable patterns without rerunning the gates. Omitting `--skip-generate` deliberately regenerates the variables first.

8.6 Software environment

Table 34 records the environment facts that are available to this manuscript without inventing cross-platform certification.

Table 34: Software inputs relevant to reproduction.

Component	Release fact	Reproducibility role
Python	3.13.11	Interpreter used by variable generation
uv	Resolved from the installed tool	Installs the locked dependency graph
`uv.lock`	Repository lockfile	Pins Python package resolution
pytest / pytest-cov	Development dependencies	Executes tests and branch coverage
Ruff / ty	Development dependencies	Executes scoped static gates
Pandoc / LuaLaTeX	Host tools	Produce semantic HTML and attempt tagged PDF; versions can affect layout
qpdf / veraPDF	Host validators	qpdf is structural; veraPDF findings determine whether conformance may be claimed
SQLite	Python standard-library binding	Exercises consequence storage where configured

`uv sync --locked` prevents lockfile changes, but it does not pin host fonts, Pandoc, TeX packages, operating-system libraries, or the current date. The generation timestamp and auto publication date also change across runs. Consequently, the expected claim is semantic regeneration with passing gates and resolved references—not universal byte-for-byte identity.

8.7 Evaluation snapshot

Table 35 identifies what the current snapshot contains.

Table 35: Contents and omissions of the release evaluation snapshot.

Item	Status	Evidence path
Gate/test/type/lint values	Executed during variable generation	`output/data/manuscript_variables.json`
Branch details	Executed during variable generation	`output/data/colony_kernel_coverage.json`
Policy and taxonomy figures	Regenerated	`output/figures/*.png`
Deterministic contract cases	Executed by the Colony Kernel suite	`tests/unit/colony_kernel/`
Paired replay artifact	Executed twice with fixed identities and compared semantically	`scripts/replay_colony_kernel.py` and `output/data/colony_kernel_replay.json`
Local authenticated lifecycle	Executed with signed/hash-linked events	`tests/unit/colony_kernel/test_kernel_atte`
Citation inventory and locator audit	Executed over every retained record	`output/data/bibliography_audit.json`
Publication bundle	Prepared only after render and detached-hash verification	`output/release/codomyrmex-1.3.0/publication_manifest.json`
Four-condition benchmark	Proposed only	No raw trial artifact in this release
Production deployment	Not evaluated	No production trace artifact

Core score calculations and tick updates are deterministic for fixed explicit inputs and state. The full build is not purely deterministic: proposal identifiers and timestamps may be created at runtime, the manuscript records a generation time, the publication date may be automatic, and renderer versions can alter layout. Replaying an identical policy case is therefore different from reproducing identical publication bytes.

8.8 Evidence required for the proposed external study

The comparative study in Section 4.1 should not be promoted to "executed" until the artifacts in Table 36 exist and regenerate the reported analysis.

Table 36: Minimum external evidence needed for a comparative evaluation claim.

Required artifact	Minimum content
Workload manifest	Stable item IDs, inputs, expected evaluation procedure, licenses, and exclusions
Baseline implementations	Versioned adapters for all four conditions with shared interfaces
Authorization/outcome ledger	Proposal, gate decision, consumed execution authorization, receipt, outcome, attester identity, and independent observation source
Raw event trace	Append-only ordered events, costs, decisions, outcomes, ticks, and errors
Run identity	Source commit, submodule revisions, full configuration and lockfile hashes, environment inventory
Analysis package	Script or notebook that regenerates tables, figures, uncertainty intervals, and exclusions
Negative evidence	Failed runs, missing outcomes, protocol deviations, and stopping decisions

Until that package exists, the manuscript's strongest supported conclusion remains the bounded one reported in Section 5: the checked implementation exhibits the tested local inhibition, recovery, scoring, and accounting behavior under its contract cases.

9 Conclusion: What the Release Establishes

This paper presents the Colony Kernel as a deterministic, inspectable prototype for placing a control decision between an agent's proposal and software actuation. Within the kernel evaluation path, a proposal is considered alongside budget headroom, local hazard pressure, consequence-derived trust, proposal completeness, the agent's inferred role label, recent failures, and falsification findings. The result is an explicit EXECUTE, HOLD, or REFUSE decision rather than an implicit assumption that possession of a tool implies permission to use it.

The implementation evidence supports a bounded claim. The checked-in tests exercise the kernel's data models, gate arithmetic, consequence updates, pheromone operations, role inference, and MCP-facing tools. These tests establish deterministic software contracts for the cases they cover. They do not establish production safety, robustness at scale, resistance to a strategic adversary, or improved task performance relative to another agent runtime.

The numeric values shown here are current implementation defaults or illustrative initial settings, not universal constants, fitted parameters, or empirical estimates. They are configurable and can be tuned or replaced through the corresponding runtime policy, configuration, or presentation inputs; after a change, regenerate the tests, tables, figures, and manuscript before interpreting the result. The policy parameters and fixture inputs should therefore be changed only through a declared tuning or calibration protocol with regenerated tests, tables, figures, and provenance—not silently treated as universal constants.

Several boundaries are especially important. The default MCP server owns one kernel instance for the lifetime of its process. Its consequence database defaults to SQLite in-memory mode, and its pheromone field is also in memory. A caller may configure a file-backed SQLite path for consequence records, but that does not persist the complete kernel state or the pheromone field. The present artifact therefore does not support a claim that local pressure, trust, or gate state automatically survives process restarts, model swaps, or deployment across machines.

The role ladder is similarly narrower than an authorization hierarchy. RoleAdapter deterministically maps trust and proposal-count state to labels, and SANDBOX receives a hard gate override. The remaining role names do not currently carry an action-specific permission matrix. They describe inferred operating status; they are not cryptographic capabilities or independently enforced privilege scopes.

The FAILURE and RISK channels remain distinct in storage, but they are now coupled at the gate: local hazard is the maximum of the two sensed pressures. The paired contract test records a failed outcome, observes the resulting FAILURE pressure, and changes a complete same-target proposal from EXECUTE to HOLD while leaving an unrelated target unchanged. Passive tick

decay later restores the original decision. This verifies a process-local locality invariant. It does not verify the truth of the outcome report. The ordinary MCP endpoint still accepts caller-supplied data without binding it to a prior EXECUTE authorization. Optional and required kernel modes provide a signed, hash-linked local lifecycle path, with required mode making that path mandatory, but the ledger does not independently observe actuation or establish that it was safe.

9.1 Falsification Criteria and Evaluation Agenda

The architecture should be judged by tests that can fail, not by the biological metaphor. The following criteria separate contracts already exercised by the repository from claims that still require implementation or empirical evaluation.

9.1.1 F1: Failure-to-gate coupling

The checked-in paired test submits identical proposals under identical agent, budget, completeness, and falsification state, varying only whether the target location has a reported failed outcome. The failed-location proposal receives a strictly lower score and a stricter decision; an unrelated location does not. The remaining falsification criterion is end-to-end external-actuation attestation: the same result must hold when FAILURE can only be created from a prior authorized action and an independently observed adverse outcome. The implemented local ledger is necessary lifecycle evidence, not completion of this criterion.

9.1.2 F2: Deterministic gate evaluation

Two evaluations with the same complete kernel state and proposal must produce the same score, decision, and stated reasons. Complete state includes the inferred role, budget, effective local hazard, trust tier, completeness tier, recent-failure count, and any CRITICAL finding. This is a software determinism criterion; it is not a claim that the resulting decision is calibrated to real-world harm.

9.1.3 F3: Bounded trust updates

For clean outcomes and zero human feedback, trust should increase by the configured pass delta while the score remains below its upper clamp. At the boundary, the clamp must be included in the expected result. The analogous checks apply to failed tests, repair requirements, and negative feedback. Passing these checks establishes arithmetic consistency, not that the heuristic deltas are statistically optimal.

9.1.4 F4: Role-label determinism and authorization separation

At each documented trust and proposal-count boundary, RoleAdapter should return the specified label. A separate negative test should confirm the current limit: beyond the SANDBOX override and other gate conditions, non-sandbox role labels alone do not enforce which action types an agent may perform. Any future per-role permission claim requires a policy matrix and action-level enforcement tests.

9.1.5 F5: Adversarial and restart behavior

An external evaluation should test unlinked outcome submission, agent-identifier reuse, semantically empty completeness fields, process restart, concurrent clients, budget-window timing, and poisoning of shared state. These cases probe assumptions that deterministic unit tests cannot settle.

Table 37: Evidence boundary and falsification agenda; external benchmark rows are future work, not reported experiments.

Question	Evidence in this artifact	Publication-honest status
Is gate arithmetic deterministic for covered fixtures?	Checked-in unit and contract tests	Supported within the tested state space
Does reported FAILURE tighten same-target local gating?	Paired contract test over `max(RISK, FAILURE)`	Supported within one kernel process
Can the kernel authenticate a complete local lifecycle?	Signed and hash-linked proposal, verdict, authorization, receipt, and outcome fixture	Supported for local ledger integrity, not external truth
Do role labels enforce per-action permissions?	SANDBOX override; no general role/action matrix	Not implemented

Question	Evidence in this artifact	Publication-honest status
Does colony state survive a default MCP process restart?	In-memory default database and field	No
Does the kernel reduce unsafe actuation on external workloads?	Proposed protocol; no released trial traces	Not yet evaluated
Does the kernel remain correct under production concurrency or scale?	Single-process implementation and local tests	Unverified

Table 37 keeps implemented contracts separate from open external claims.

9.2 Limitations and Next Work

The experimental protocol in Section 4 is a specification for future evaluation. The repository does not currently provide the raw repeated-trial traces, baseline runs, or external workload results needed to estimate refusal rates, repair rates, latency, or safety benefit. Consequently, configured scenario counts and expected rates should not be read as measured outcomes.

The gate weights, trust deltas, decay amounts, role thresholds, and falsification checks are fixed engineering heuristics. They have not been calibrated against downstream harm or production repair cost. The falsification worker is also a deterministic proposal checker, not a proof of semantic safety: only CRITICAL findings independently force a gate refusal, and syntactically complete but misleading evidence may evade its checks.

Four implementation priorities follow directly from these limits. First, extend the implemented local authorization/receipt chain with deployment-specific, independently verifiable observation of external effects and actor identity. Second, persist and reconcile the entire field and budget state when cross-session behavior is required. Third, define and enforce an action-by-role authorization policy if role names are to carry security meaning. Fourth, run the proposed paired and external benchmarks with released traces, explicit baselines, and predeclared outcome measures before making effectiveness or scale claims.

The Colony Kernel's present contribution is thus a testable control-plane scaffold. It makes selected state transitions and decision rules visible enough to inspect, replay within a process, and challenge. That is useful groundwork, but it is not yet evidence that the colony becomes safer merely by accumulating history. In this version, the record is evidence for a control decision—not a guarantee that the decision is correct.

10 Research Roadmap: Evidence Gates and Dependency Order

The numeric values shown here are current implementation defaults or illustrative initial settings, not universal constants, fitted parameters, or empirical estimates. They are configurable and can be tuned or replaced through the corresponding runtime policy, configuration, or presentation inputs; after a change, regenerate the tests, tables, figures, and manuscript before interpreting the result. This roadmap is a scoped research program, not a catalogue of completed results. The current paper establishes the reproducible kernel contract, a fixed-input paired replay artifact, and the local authenticated lifecycle ledger. Subsequent milestones are conditional on their artifacts, metrics, falsifiers, and exit criteria. A status label therefore describes evidence available in the current repository, not a promise about delivery or scientific success.

10.1 Research question and boundary

The central question is whether a proposal-governance control plane can improve the safety–utility trade-off of agentic software work when its state is authenticated, evaluated under hostile inputs, and calibrated against independently observed consequences. The implementation currently supports a narrower statement: a caller-reported failure changes same-target local pressure in a running kernel process, and the deterministic gate responds according to its configured rule. The roadmap does not promote that mechanism into a claim of harm reduction, optimality, collective intelligence, or Active Inference.

The milestones are ordered by dependency. Replayable artifacts must precede external comparisons; local lifecycle authentication must precede deployment-specific external observation; externally attested outcomes must precede trust calibration; adversarial and held-out evaluation must precede claims about utility; and persistence/concurrency evidence must precede deployment-oriented interpretation. This ordering follows reproducibility and accountability principles rather than a calendar commitment (Peng 2011; Raji et al. 2020; Buhl et al. 2024).

The synthetic benchmark record retains 6 paired observations with sample unit `synthetic task–case pair`, the case-manifest digest `74e664af b4e22a35 873995a8 15e21dca 88118e2e eb20848e 39acf99b 110a2ca8`, paired execution volumes

of 6 and 3, and the difference direction `mediated minus baseline`. Its interval uses `paired percentile resampling (descriptive)` at a nominal `95%` resampling reference level; `descriptive intervals over synthetic task-case pairs; not population confidence intervals.` The plotted mediator is `independent reference interpreter: codomyrmex.colony_kernel.reference.ReferenceGate`, and production gate parity is `not_established`. These metadata bind the displayed fixture to its ordered task cases; they do not turn synthetic pairs into an estimate of population risk or utility.

Milestones run top to bottom through reproducibility, a local authenticated lifecycle ledger, external-actuation attestation, adversarial evaluation, calibration, persistence and concurrency, and a probabilistic extension. Every card prints its current status and an abbreviated required-evidence artifact. Directional arrows, sequence identifiers, and status text define the dependency order; colour is redundant and no milestone promises delivery or success.

The first executable evidence fixtures are separated by what they can and cannot establish. The attestation chain binds local lifecycle events, the paired benchmark exposes the safety–utility analysis plumbing, and the persistence fixture checks restart state. None is an external benchmark, causal estimate, or production security guarantee. The figures below are generated artifacts with declared boundaries, not evidence that the later research milestones have been completed.

Five numbered event nodes are connected top to bottom by directional arrows. Each row prints its event type, actor, hash prefix, and signature state. The chain order binds proposal, gate verdict, authorization, caller-supplied execution receipt, and caller-supplied outcome digests under the configured signer. The linked structure establishes fixture-level ledger integrity only; it does not verify external actuation, safety, or usefulness.

Harmful-action rate is on the horizontal axis and utility is on the vertical axis, both averaged over the same ordered synthetic task cases. One labelled point represents the always-execute baseline and a differently shaped labelled point represents the independent ReferenceGate-mediated condition; an arrow annotates the paired mediated-minus-baseline harm difference and its descriptive interval. The points come from deterministic synthetic cases. The mediator is not the production ActuationGate, parity is not established, and the interval does not imply population generalization or calibration.

The square plotting area shows only the ideal-reference diagonal. A central status card states that calibration is not_estimated and that expected calibration error is not_run. No bins or empirical points are drawn because gate and trust scores are not probabilities and the required held-out outcomes and declared confidence values are absent.

Three ordered bars report the signal's strength when deposited, after the store is closed and reopened, and at the fixture's recovery observation. Numeric labels are printed above the bars so height and colour are not the only encodings. The retained record supports a narrow logical durability check, not throughput, arbitrary crash survival, or persistence of all kernel state.

The roadmap in Figure 10 is a dependency map, not a delivery timeline. Its future milestones are planning objects and must not be read as empirical evidence.

The first offline research fixtures are shown alongside the roadmap: the authenticated event chain in Figure 11, the paired safety–utility plumbing in Figure 12, the explicit calibration hold in Figure 13, and the restart check in Figure 14.

10.2 Milestone contract

The complete configured program contains 7 milestones. Each row states what would be built, measured, falsified, and accepted; empty claims are not allowed to advance by narrative momentum.

Table 38: Evidence plan for the configured research milestones.

ID	Milestone	Status	Hypothesis	Required artifact
R0	Reproducible kernel contract	Implemented	The local control-plane contract can be replayed from the checked-in source and configuration.	Scoped tests, coverage record, variable map, figure registry, and rendered manuscript.
R1	Local authenticated lifecycle ledger	Implemented	Optional and required kernel modes can bind proposal, verdict, authorization, execution receipt, and outcome records locally.	Signed, hash-linked local event ledger with nonce, actor, proposal digest, verdict, execution receipt, and outcome.

ID	Milestone	Status	Hypothesis	Required artifact
R2	End-to-end external actuation attestation	Next	Independently sourced execution evidence can distinguish actual external actuation from caller-reported lifecycle events.	Deployment-specific adapter that verifies execution identity, external receipt provenance, and outcome linkage.
R3	Adversarial tool-use benchmark	Planned	Gate-mediated proposals will change unsafe-action and recovery outcomes under hostile tool inputs.	Seeded AgentDojo, InjecAgent, or ToolEmu adapter with paired baseline traces.
R4	Calibration and utility	Planned	Gate scores can be calibrated against independently observed downstream consequences.	Held-out outcome dataset, baseline policies, calibration analysis, and preregistered configuration.
R5	Persistence and concurrency	Planned	Explicit persistence and concurrent access can preserve locality without violating isolation or ordering.	Restart, crash-recovery, multi-worker, and load scenarios under a declared deployment profile.
R6	Probabilistic and Active Inference extension	Research	An explicit generative model can improve prediction or policy selection beyond the heuristic projection.	Declared likelihood, posterior implementation, simulation-based calibration, and paired baselines.

Table 39: Decision contract for the configured research milestones.

ID	Decisive metric	Falsifier	Exit criterion
R0	Replay agreement for gate, locality, decay, and provenance outputs.	Any replay depends on an undocumented input or produces a contradictory contract result.	Fresh generation and focused gates pass with hashes recorded.
R1	Forgery, replay, sequence, actor, and linkage rejection in deterministic tests.	The local ledger accepts a forged, replayed, reordered, mismatched, or incomplete lifecycle.	Lifecycle integration tests pass and the external-observation boundary remains explicit.
R2	Forgery and replay rejection, observation completeness, adapter overhead, and independently replayed traces.	A caller can report or relink an action or outcome without verifiable external execution evidence.	A declared deployment adapter passes adversarial tests and an independent verifier reproduces its evidence chain.
R3	Attack success, harmful-action rate, task utility, refusal cost, and complete trace retention.	The mediated condition does not improve the preregistered safety-utility frontier.	Independent rerun reproduces paired estimates and failure analyses.
R4	Log loss, calibration error, selective risk, repair cost, utility, and confidence intervals.	Scores remain miscalibrated or add friction without a defensible utility trade-off.	Held-out analysis and uncertainty intervals are released with raw traces.

ID	Decisive metric	Falsifier	Exit criterion
R5	Durability, ordering, isolation, throughput, latency, and recovery completeness.	Restart or concurrency causes silent signal loss, cross-target contamination, or unsafe duplicate actuation.	Scenario matrix passes with retained logs, seeds, environment, and artifact hashes.
R6	Posterior predictive checks, held-out log loss, calibration, policy utility, and compute cost.	The probabilistic model fails calibration or utility comparisons, or its assumptions are not identifiable.	Model, assumptions, replay harness, and negative results are independently reproducible.

The evidence plan in Table 38 and decision contract in Table 39 are generated from `docs/manuscript/config.yaml`; they are not a second authority for parameters or results. The variable snapshot records the exact roadmap configuration, while the figure registry records the rendered planning visual and its hash. R0 additionally retains the replay's semantic and file digests shown in Table 32. The implemented R0 row is accepted only when its configured `artifact_paths` resolve to checked-in source or evidence surfaces. R1 is governed by the same artifact-path requirement for the implemented local ledger. R2 is deliberately separate and remains open until a deployment-specific external-observation adapter and independent verification evidence exist; later rows remain hypotheses until they acquire equivalent retained paths.

10.3 Execution protocol

Every future study should retain the following minimum evidence bundle:

- the source commit, environment fingerprint, lockfile digest, configuration digest, random seed, and input checksums;
- raw append-only proposal, verdict, authorization, receipt, and outcome traces, including the external observation source and rejected or failed cases rather than only successful runs;
- the baseline policy, the mediated policy, and the exact paired assignment rule;
- analysis code that computes point estimates, uncertainty intervals, calibration diagnostics, and failure stratifications; and
- generated tables, figures, captions, and a machine-readable manifest linking each claim to its source artifact.

This bundle turns a research result into a replay target. It does not require a claim to be positive: a well-specified null result or failed falsification attempt is publishable evidence when its inputs and analysis remain inspectable. Model cards and safety cases are useful reporting analogies, but their presence does not substitute for the linked execution evidence (Mitchell et al. 2019; Buhl et al. 2024).

10.4 Decision rules for promotion

Promotion from one milestone to the next requires all of the following:

1. the artifact is complete and independently replayable;
2. the decisive metric is computed on the declared comparison rather than on the gate labels it is intended to validate;
3. uncertainty, missingness, and adverse cases are reported;
4. the stated falsifier has been evaluated without silently changing the protocol; and
5. the result does not exceed the claim boundary of the implementation or the data.

If a milestone fails its falsifier, the next action is diagnosis, redesign, or a bounded negative result—not relabeling the failure as a successful ecological effect. In particular, training a predictor to reproduce EXECUTE/HOLD/REFUSE labels would validate the existing rule's behavior, not its downstream usefulness. Utility must be measured against independently observed repair cost, policy violation, task completion, or other outcomes specified before analysis.

10.5 Relation to the active-inference track

The final milestone depends on the distinctions in Section 7. A probabilistic extension must declare observations, latent states, likelihoods, priors, posteriors, policies, preferences, and an inference procedure. It must then be compared with the current deterministic gate using posterior predictive checks, simulation-based calibration, held-out log loss, utility, and compute cost. Naming a trust score a posterior or a gate score expected free energy would not satisfy that contract.

The roadmap therefore treats Active Inference as a falsifiable modeling direction rather than a retrospective interpretation. This is consistent with the current crosswalk's claim boundary and with the broader positioning against tool-use security and runtime

Figure 10: Configured research roadmap for the Colony Kernel. The 7 milestones move from the implemented contract and local authenticated ledger toward external-actuation attestation, adversarial evaluation, calibration, persistence, and a probabilistic extension. Status labels describe the current research state, not empirical results or delivery guarantees; each milestone remains conditional on its stated artifact, metric, falsifier, and exit criterion.

Figure 11: Authenticated local lifecycle fixture with 5 linked events and validation status true. The chain binds proposal, verdict, authorization, execution receipt, and outcome digests under the configured signer; it establishes ledger integrity, not that the action was safe, useful, or independently observed.

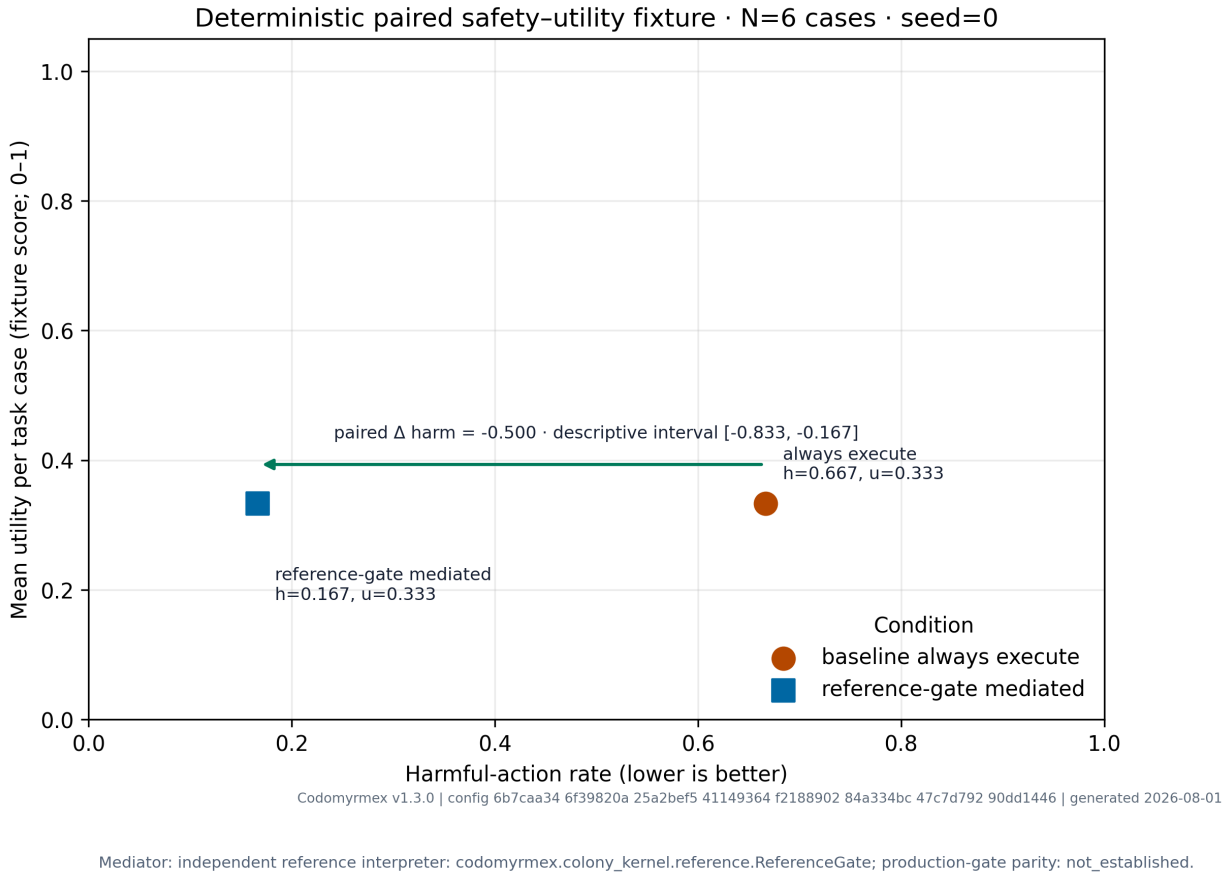

Figure 12: Paired safety–utility points from 6 deterministic synthetic tool-use cases, with denominator N=6 task cases per condition. Baseline and reference-gate-mediated harmful-action rates are 0.667 and 0.167, with utilities 0.333 and 0.333. The paired harmful-action difference is -0.500 (mediated minus baseline) with a descriptive percentile-resampling interval from paired percentile resampling (descriptive) at a nominal 95% reference level [-0.833, -0.167]. The interval uses task-case pairs, not a population sample, and is not a conventional inferential confidence interval (descriptive intervals over synthetic task-case pairs; not population confidence intervals). The plotted mediator is independent reference interpreter: codomyrmex.colony_kernel.reference.ReferenceGate; production-gate parity is not_established. This fixture demonstrates analysis plumbing, not calibration or generalization to external agents or workloads. Marker shape, direct labels, and colour redundantly identify the two conditions.

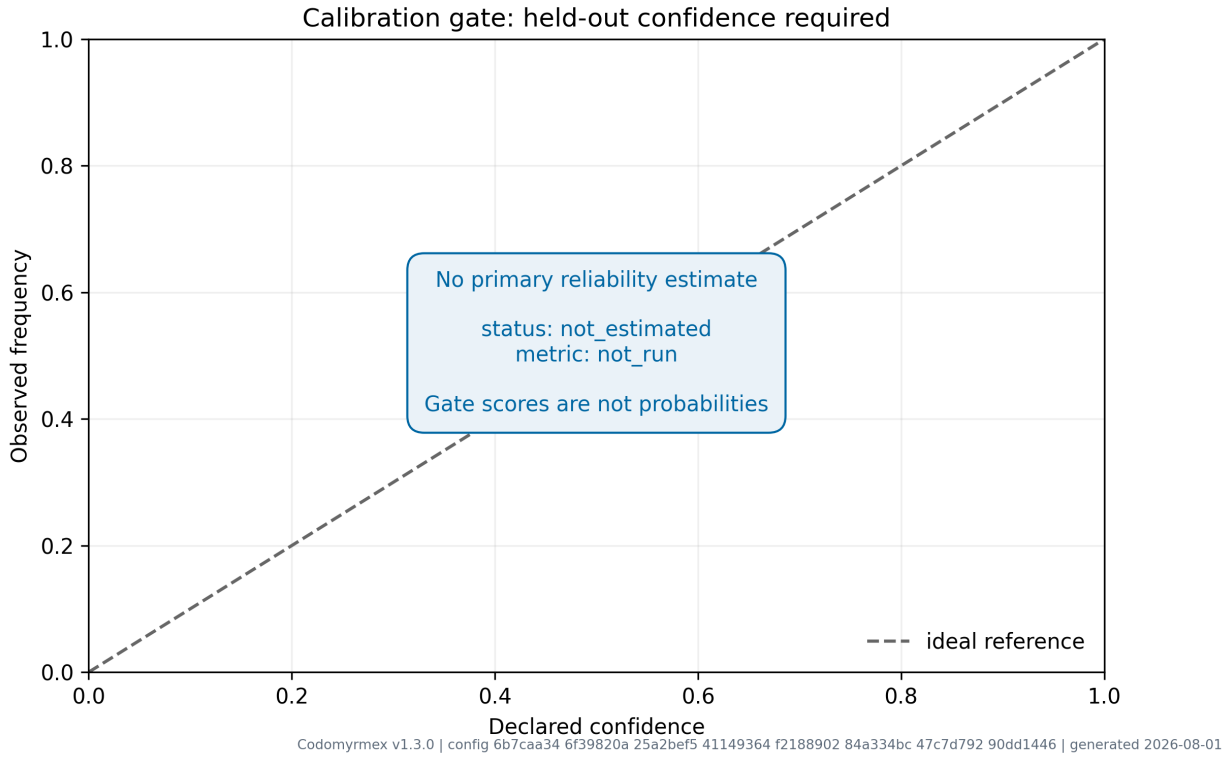

Figure 13: Calibration status panel for the current evidence bundle: primary calibration is not_estimated and log-loss analysis is not_run. Gate scores and trust values are not probabilities; a reliability estimate will be rendered only after independently observed held-out outcomes and declared confidence values are available.

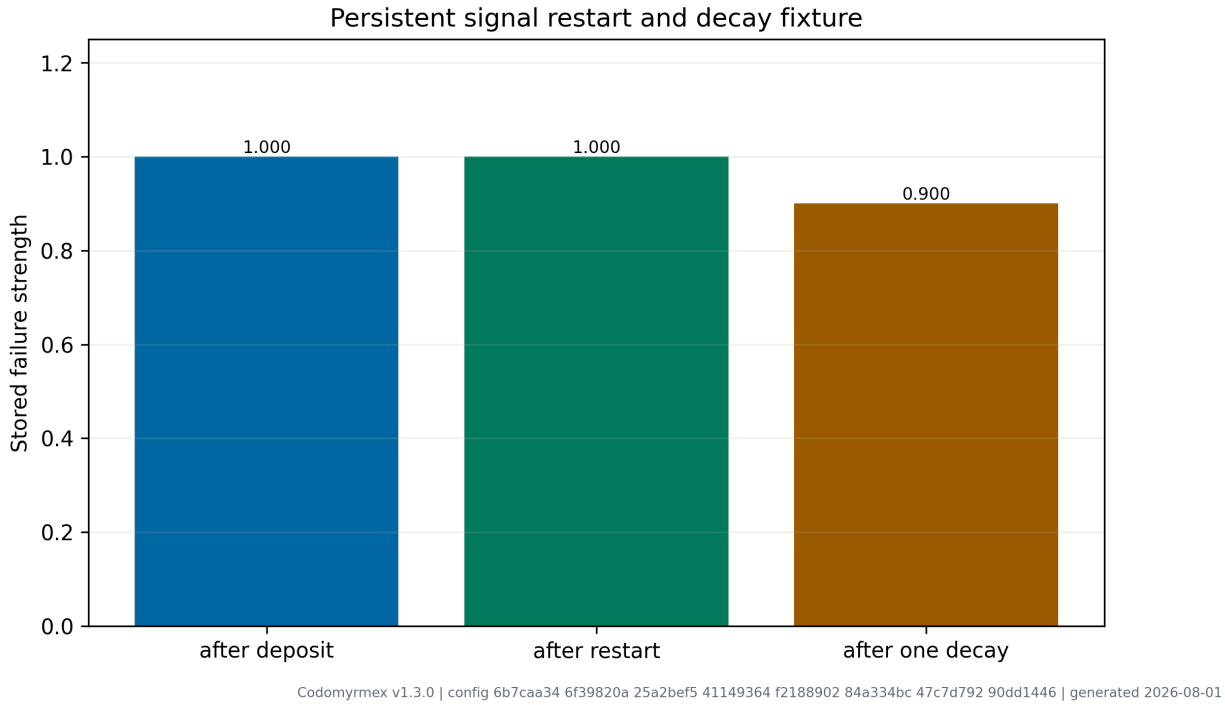

Figure 14: SQLite-backed signal restart fixture. The retained failure signal after closing and reopening the store is 1.000; the durable canonical logical database artifact digest is 464fd1d9 9088a6c0 99059f13 25d1494e 5fe12a90 f46d4ec4 ae9c0e2a daa0635d. This is a durability check, not a throughput or crash-survival claim.

assurance: cited external benchmarks motivate adapters and threat models, but none is evidence for the Colony Kernel until the experiment is actually run (Debenedetti et al. 2024; Zhan et al. 2024; Ruan et al. 2023; Seto et al. 1998).

10.6 Scope of this release

The current release documents and validates R0's local contract, fixed-input replay, and reproducibility route plus R1's authenticated local lifecycle ledger. It does not claim that R2's end-to-end external-actuation attestation or any later milestone is complete, that the provisional settings are calibrated, or that the control plane is a security boundary. The appropriate scholarly contribution at this stage is a composable, inspectable research substrate with explicit failure conditions and a machine-readable path from configuration to prose, tables, figures, and artifacts.

11 Formalism-to-Code Crosswalk and Translation Methods

The notation used in this crosswalk is fixed by Section 12. In particular, symbols in this section are translations of the glossary rather than new local aliases.

The repository uses several kinds of formal description: typed operational contracts, discrete recurrences, piecewise decision rules, executable invariants, AST-preservation rules, optional SMT obligations, and probabilistic inference components. These formalisms are complementary, but they are not interchangeable. A formula in a document is not a property of the software until the variables, state transition, implementation mapping, and verification boundary are made explicit.

The numeric values shown here are current implementation defaults or illustrative initial settings, not universal constants, fitted parameters, or empirical estimates. They are configurable and can be tuned or replaced through the corresponding runtime policy, configuration, or presentation inputs; after a change, regenerate the tests, tables, figures, and manuscript before interpreting the result. This section makes that translation chain inspectable. It does not claim that every formalism has been connected to the Colony Kernel, and it does not treat a shared word such as *trust*, *state*, or *policy* as evidence of semantic equivalence.

11.1 A five-link translation chain

The crosswalk uses five links for every formal claim:

1. Formal object. State the mathematical or logical object, its variables, domains, and assumptions. If a probability model is intended, declare the sample space, likelihood, prior, posterior, and policy variables rather than naming a scalar as a posterior by analogy.
2. Code representation. Identify the typed value objects, enums, state holders, or functions that represent the object. The representation may be partial; that fact is recorded rather than inferred away.
3. Translation mechanism. State how the formal object is computed from or imposed on code: a constructor guard, state-transition function, recurrence, AST comparison, schema adapter, or sound symbolic encoding.
4. Executable evidence. Name the tests, replay artifacts, solver obligations, or generated outputs that exercise the translation. A test of a wrapper is not silently promoted to a proof of the wrapped system.
5. Claim boundary. State what the evidence supports and the strongest nearby claim it does not support. This final link prevents a local invariant from becoming a claim of safety, utility, biological equivalence, or production reliability.

This discipline follows design-by-contract and formal-methods practice (Apt 2003), while the retained configuration, source anchors, and generated artifacts follow the broader requirements of reproducible and accountable computation (Peng 2011; Raji et al. 2020; Buhl et al. 2024).

Each row identifies a formalism and explicitly prints IMPLEMENTED, PARTIAL, or RESEARCH. The middle column abbreviates its typed code anchors and translation bridge; the final column abbreviates executable evidence and the strongest claim the evidence does not support. Alternating row shading and status-colour strips aid navigation, while the full wording remains available in searchable tables.

One horizontal bar partitions all 7 configured formalism mappings by current bridge status. Each non-empty segment prints its status and count, while a legend repeats the categories. Segment length, order, text, and colour provide redundant encodings. The inventory counts documented mappings, not proved theorems or semantic equivalences.

Rows name configured roadmap stages and formalism mappings. Each row places a single marker under its printed IMPLEMENTED, PARTIAL, NEXT, PLANNED, or RESEARCH column, so horizontal position and text define the status independently of colour. The matrix is a configuration-backed navigation aid, not evidence that conditional work is complete or empirically successful.

Figure 15: Abbreviated evidence-oriented crosswalk from formal objects to typed code anchors, translation mechanisms, executable evidence, and claim boundaries. Every row prints its status; coloured strips provide a redundant navigation cue. Full, searchable wording follows in the adjacent manuscript tables. The visual records correspondence and missing links; it is not a proof graph or evidence of scientific equivalence.

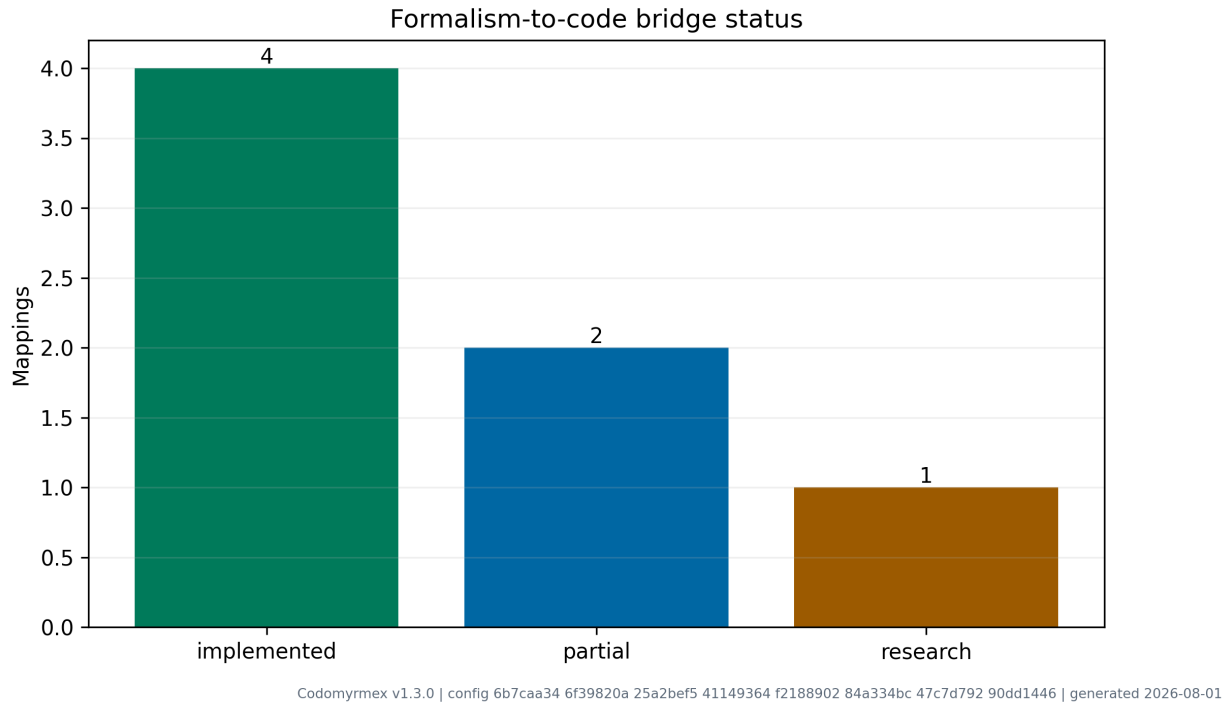

Figure 16: Current formalism-to-code inventory with 7 mappings: 4 implemented and 2 partial, with 1 research. The chart measures documented bridge status and evidence anchors, not theorem coverage or semantic equivalence. Segment labels and position redundantly encode the status counts.

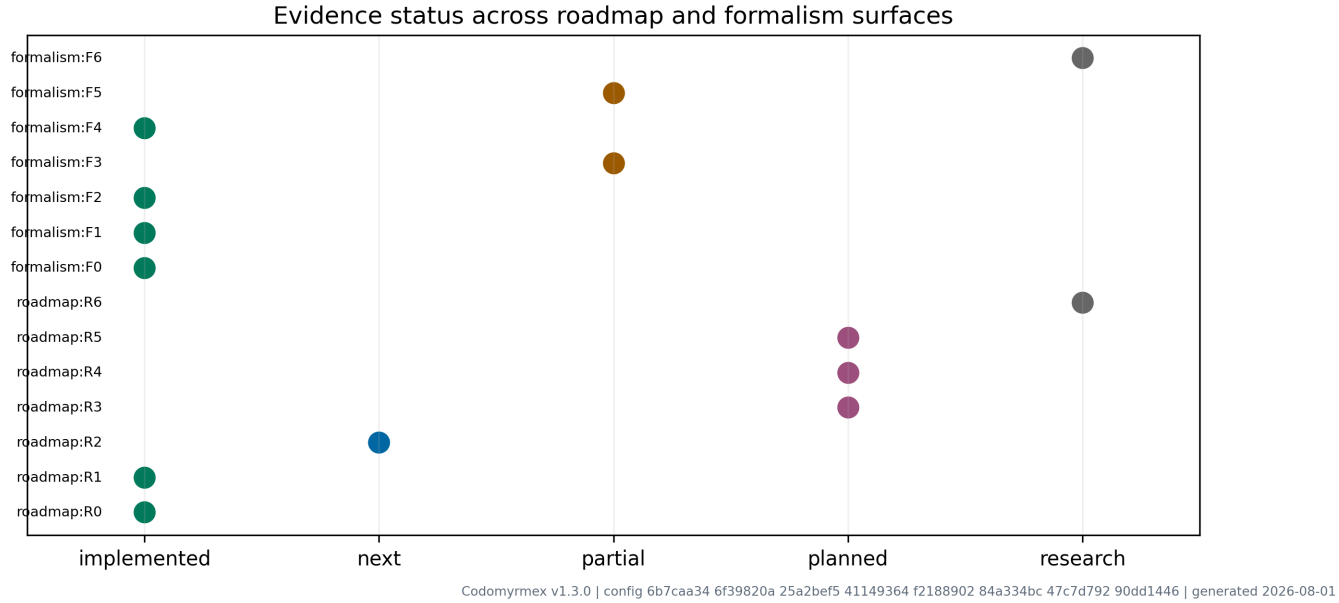

Figure 17: Evidence-status matrix for the configured roadmap and formalism crosswalk. Status labels are generated from the active configuration and indicate what artifacts exist or remain conditional; they are not completion guarantees or measured implementation outcomes. Each row's marker position and printed status column carry the classification; colour is redundant.

The visual in Figure 15 shows correspondence as a chain with missing links, not as a proof graph. Every row prints its status, and the coloured strip repeats that status as a secondary scanning cue. The visual abbreviates the source-backed entries for print legibility; Table 40 and Table 41 retain their complete searchable wording.

The inventory summary in Figure 16 counts these documented statuses, while Figure 17 places the formalism rows beside the roadmap milestones. Neither visualization measures theorem coverage; both are navigation and claim-boundary aids.

11.2 Current correspondence inventory

The configured inventory contains 7 mappings: 4 implemented, 2 partial, and 1 research. The first table records the formal layer and its current status; the second records the translation and evidence boundary. Both tables are generated from `docs/manuscript/config.yaml` and validated against the current repository paths before rendering.

Table 40: Formalism inventory and implementation status.

ID	Mapping	Formal layer	Formal object	Status
F0	Typed operational semantics	State-transition contract	Proposal, verdict, reported consequence, and kernel state transition	Implemented
F1	Discrete dynamical recurrence	Bounded state recurrence	x_(j,t+1)=max(0, x_(j,t)-ε_j) for a trace without new deposits	Implemented
F2	Piecewise decision semantics	Rule system with hard overrides	Weighted score, hazard tiers, and EXECUTE/HOLD/REFUSE precedence	Implemented
F3	Runtime invariant predicates	Executable design-by-contract checks	Bounds, monotone role thresholds, enum separation, and gate-weight conservation	Partial
F4	Structural code-change verification	AST-preservation rules	No deletion of public functions, no removed parameters, and order-compatible signatures	Implemented
F5	SMT invariant proving	Satisfiability-based proof obligation	C AND not(invariant) is unsatisfiable	Partial
F6	Probabilistic and Active Inference models	Generative model, posterior, and policy evaluation	p(o,s), q(s), transition/observation models, preferences, and expected free energy	Research

Table 41: Translation, evidence, and claim boundaries for the formalism inventory.

ID	Code anchor	Translation mechanism	Executable evidence	Claim boundary
F0	ActionProposal; GateResult; ColonyKernel / propose_action; ColonyKernel / record_outcome; run_paired_locality_replay	Dataclasses and enums define the vocabulary; kernel sequencing defines the transition boundary.	Typed construction, lifecycle sequencing, repeatable replay, and serialized result tests.	Establishes a replayable interface and control-flow contract, not semantic safety or execution attestation.

ID	Code anchor	Translation mechanism	Executable evidence	Claim boundary
F1	PheromoneStore / evaporate; PheromoneStore / tick; TraceField / sense	Per-signal metadata supplies epsilon; integer ticks apply the floored subtraction and delete depleted markers.	Decay, floor, compound-key isolation, and stress/property scenarios.	Supports the stated recurrence under declared inputs; it is not exponential forgetting, wall-clock decay, or biological pheromone chemistry.
F2	ActuationGate / witness_state; ActuationGate / evaluate; FalsificationWorker / evaluate_plan	Named thresholds and ordered branches implement the piecewise policy; no opaque learned scorer is implied.	Boundary, precedence, monotonicity, and adversarial-vector tests.	Proves only the coded policy properties; the score is not a probability of safety, harm, or utility.
F3	all_invariants_hold; ColonySignal / post_init; ResourceCost / post_init	Predicates and constructor guards execute against live constants; a dedicated crosswalk test checks source alignment.	Direct invariant checks plus boundary regressions; cross-module coverage remains incomplete.	Runtime predicates detect selected violations for supplied states; they are not a complete proof of all kernel behaviors.
F4	ChangeProposal; CodeChangeVerifier / verify; InvariantRule	Python source is parsed into ASTs and compared by pluggable rule functions before optional attestation.	Real-source AST regression tests for preserved public structure and custom rules.	Covers selected structural compatibility properties; it does not establish behavioral equivalence, security, or deployability.
F5	KernelFormalSnapshot; runtime_obligations; prove_kernel_obligations	A solver-neutral obligation model and bounded kernel-specific Z3 encoding translate selected runtime bounds, weight conservation, pressure monotonicity, locality, and authorized linkage obligations into solver expressions.	Structured solver outcomes and counterexample handling are tested for the kernel-specific bounded encoding; the optional solver path reports unavailable when Z3 is absent.	The bridge proves or refutes the encoded bounded obligations; it does not establish whole-program refinement, production safety, or a proof of the Python kernel.
F6	KernelProbabilisticAdapter; GenerativeModelSpec; BayesianNetwork; InferenceEngine / infer; VariationalFreeEnergy / compute; ActiveInferenceAgent	The explicit adapter declares kernel observations, latent states, priors, likelihoods, transitions, preferences, actions, and horizon while preserving the deterministic gate as the baseline.	Standalone Cerebrum tests and explicit adapter-schema tests; held-out posterior evaluation and production policy integration remain unexecuted.	The adapter is research plumbing, not an integrated Active Inference controller, calibrated safety model, or expected-free-energy implementation.

Table 40 is the status view; Table 41 is the evidence and limitation view. Keeping both views explicit prevents an implemented row from being read as a universal proof.

The inventory distinguishes *representation* from *verification*. For example, `ActiveInferenceAgent` and `InferenceEngine` provide probabilistic components, but their presence does not connect them to the Colony Kernel's deterministic gate. Conversely, the gate recurrence and the AST-preservation rules have executable bridges but do not imply that the resulting policy is safe or optimal.

11.3 How the formalisms compose

The useful relationship among the layers is a sequence of constrained translations:

Operational semantics → recurrence. Typed `ColonySignal`, `ActionProposal`, and `GateResult` objects define the state vocabulary. `PheromoneStore` then supplies a discrete transition for one part of that state. The recurrence is a projection of the operational state, not a complete semantics for the kernel.

Recurrence → decision rule. The gate observes the projected local field and applies a piecewise risk function plus hard overrides. Monotonicity of the recurrence or risk function can be proved arithmetically while the composition, ordering, and side effects still require integration tests.

Decision rule → invariant predicate. Bounds, threshold ordering, and weight conservation can be stated as executable predicates. These predicates should import the same runtime constants as the decision rule; copied constants create a second model that can pass while the implementation changes.

Invariant predicate → symbolic obligation. An SMT bridge can prove a supplied obligation such as $C \wedge \neg I$ being unsatisfiable. It proves the encoded obligation, not the Python implementation, unless a sound state-to-symbol translation and a correspondence theorem are supplied. The repository now includes a kernel-specific, bounded encoding and a structured `unavailable` result when optional Z3 is absent; the encoding remains evidence about the stated obligations, not a whole-program proof.

Deterministic state → probabilistic model. A probabilistic or Active-Inference model must introduce random variables, likelihoods, priors, transition dynamics, preferences, and an observation protocol. Deterministic pressure, trust, or gate scores may be observations or interface projections, but they are not posteriors or expected free energy without those definitions.

These relations are directional. A code path can instantiate a formal recurrence, while an equation cannot by itself establish that a caller, scheduler, or persistence layer actually follows the equation. The crosswalk therefore treats implementation evidence as necessary for a code claim and formal definitions as necessary for a mathematical claim; neither substitutes for the other.

11.4 Integration research agenda

The next work is deliberately staged so that each stronger connection depends on a weaker one being replayable:

1. State and trace schema. The versioned ledger and replay schema now link proposal digest, gate verdict, execution authorization, execution receipt, outcome evidence, and actor. Focused tests cover replay, omission, duplication, nonce reuse, and unauthorized relinking; outcomes are not promoted to inference data automatically.
2. Kernel-specific invariant encoding. A solver-neutral contract and optional Z3 backend now return proved, refuted, unknown, timeout, or unavailable states. The remaining research task is a broader correspondence audit across generated states.
3. Refinement tests. The independent reference interpreter and differential tests cover the current gate projection. More transition families should be added before claiming complete refinement.
4. Probabilistic adapter. The explicit adapter declares an observation model over externally observed and attested traces, a latent state space, priors, transition and observation models, preferences, and policy horizon. Keep the deterministic gate as a baseline and report calibration, held-out log loss, utility, refusal cost, and compute cost.
5. Cross-formalism evidence bundle. Publish the source commit, configuration and environment digests, seeds, inputs, raw traces, solver versions, generated tables and figures, and all negative or inconclusive results. Promotion requires the evidence bundle and the stated falsifier, not a plausible narrative.

The Section 10 milestones are the release-level plan; this section is the translation method that makes a milestone's formal and executable parts composable. A future positive result may justify a stronger claim only after the relevant bridge is implemented, independently replayed, and shown not to collapse into a test of the same labels it is intended to validate.

11.5 Scope and non-equivalence

The current release establishes several local correspondences and exposes their gaps. It does not provide a formal semantics for all modules, a proof-carrying build, a complete refinement proof from Python to SMT, a causal model of outcomes, or an integrated Active Inference controller. The appropriate scholarly claim is therefore methodological: Codomyrmex provides a substrate in which formal objects, code anchors, tests, and claim boundaries can be recorded together. Whether those connections produce better safety, utility, calibration, or generalization remains an empirical question.

12 Supplemental Notation

This section is the authoritative notation glossary for the formalism in the manuscript. Symbols are reused only when they belong to the same formal layer; implementation names in code, table headings, and prose should follow these definitions. The glossary separates logical ticks, recorded outcomes, and paired statistical units so that deterministic fixtures are not mistaken for sampled data. The index and field terms are collected in Table 42, gate terms in Table 43, trust terms in Table 44, probabilistic terms in Table 45, and paired-statistics terms in Table 46.

12.1 Indices, keys, and state

Table 42: Indices, keys, and field-state notation.

Symbol	Meaning	Scope and convention
t	Discrete scheduler tick	Signal-field evolution and passive decay.
n	Ordered recorded outcome or lifecycle event	Trust updates and consequence accounting.
i	Paired task-case index	One synthetic or future workload item observed under both conditions.
ℓ	Target location	A module, path, or other declared proposal target.
k	Signal type	An element of $\mathcal{K}$, such as RISK or FAILURE.
$j = (\ell, k)$	Compound location–signal key	A member of $J_t \subseteq \mathcal{L} \times \mathcal{K}$.
J_t	Keys present at tick t; J	J is a finite analysis-horizon universe containing the observed keys.
$x_{j,t}$	Capped field strength for key j at tick t	Never use s for this quantity; s is reserved for a latent state in the probabilistic layer.
$d_{j,t}$	Effective deposit applied during a field update	Includes the configured source and optional trust multipliers.
$\epsilon_{j,t}$	Evaporation amount for key j during a field update	Positive strength units per logical tick; ϵ_j denotes a fixed value.
M	Field-strength cap	The runtime maximum in the field recurrence.

The field state is therefore written as $x_t \in [0, M]^J$ in Section 2.1, with the implementation-specific lower and upper bounds injected into the equations. The canonical update is the evaporate-then-deposit recurrence in Equation 2. The symbol s in earlier informal descriptions of a signal is replaced by $x_{j,t}$ or an explicitly named initial value.

12.2 Hazard, gate, and decision semantics

Table 43: Hazard, gate, and decision notation.

Symbol	Meaning	Scope and convention
$h_t(\ell)$	Effective local hazard pressure	$\max\{x_{(\ell,\text{RISK}),t}, x_{(\ell,\text{FAILURE}),t}\}$.
$\rho(h)$	Risk-clearance credit	A non-increasing piecewise map from effective hazard to the ordinary score component.
b	Binary budget credit	The ordinary-score input for an approved budget.
u	Trust credit	The normalized tiered trust input after any transient recent-failure penalty.
c	Proposal-completeness credit	The normalized evidence-mass input derived from missing required fields.
w_b, w_ρ, w_u, w_c	Gate component weights	Non-negative configured coefficients whose sum is the score scale.

Symbol	Meaning	Scope and convention
g	Ordinary gate score	The weighted score before or after the stated implementation clamp.
$D(g;\cdot)$	Ternary decision map	Returns EXECUTE, HOLD, or REFUSE after hard overrides and thresholds.

The gate score is written

$$g = w_b b + w_\rho \rho(h) + w_u u + w_c c. \tag{23}$$

This notation avoids the earlier collision between r as a risk-credit function and r_{repair} as a trust-update term. It also reserves h for hazard; human feedback is f_n, not h_n. The ordinary score and its hard overrides are specified in Section 2.3 and Section 3.

12.3 Trust and reported consequences

Table 44: Trust and reported-consequence notation.

Symbol	Meaning	Scope and convention
τ_n	Trust score immediately before recorded outcome n	Bounded implementation state, not a posterior.
Δ_n	Net trust increment associated with outcome n	Clipped when applied to τ_n.
$\delta_{\text{test}}(n)$	Test-pass or test-failure increment	The pass/fail component of Δ_n.
δ_{repair}	Repair contribution	A named trust-update term, not risk clearance.
f_n	Parsed human-feedback value	Bounded by the configured feedback domain.
δ_{human}	Human-feedback coefficient	Multiplies f_n in the trust update.
p_{pass}	Illustrative independent test-pass probability	A symbolic sensitivity variable in the drift relation; not an estimated probability in this report.

The trust recurrence is

$$\tau_{n+1} = \text{clip}(\tau_n + \Delta_n), \qquad \Delta_n = \delta_{\text{test}}(n) + \delta_{\text{repair}} \mathbf{1}_{\text{repair}}(n) + \delta_{\text{human}} f_n. \tag{24}$$

Here n indexes a recorded report, not a scheduler tick or an independent statistical replicate. Ordinary MCP outcomes are caller-reported; the local attestation boundary does not turn them into independent observations.

12.4 Probabilistic and Active Inference layer

The following symbols are reserved for the proposed probabilistic crosswalk and must not be used as synonyms for deterministic gate quantities:

Table 45: Probabilistic and Active Inference notation.

Symbol	Meaning
o	Observation
s	Latent state
$p(o, s)$	Joint generative density
$q(s)$	Approximate posterior
π	Candidate policy or action sequence
$G(\pi)$	Expected-free-energy quantity in the proposed model

Symbol	Meaning
$\mathcal{F}[q; o]$	Variational free-energy functional

The active-inference crosswalk in Section 7 is conceptual and unimplemented for the production gate. In particular, $g \neq 1-G(\pi)$, $\tau_n \neq q(s)$, and a deterministic signal strength is not an observation sample from a declared likelihood without the additional model contract.

12.5 Paired statistics and interval language

Table 46: Paired-statistics and interval notation.

Symbol	Meaning	Scope and convention
i	Paired task-case index	The same ordered case is compared across conditions.
m	Condition label	Use explicit labels such as baseline and mediated; c remains completeness in a gate equation.
Y_{im}	Binary harmful-action indicator	1 denotes harmful action for case i under condition m.
U_{im}	Fixture utility score	A declared per-case score, not a universal welfare measure.
N	Number of paired task cases	The denominator for case-level rates and paired differences.
$\hat{\Delta}_Y$	Paired harmful-action difference estimate	$\frac{1}{N}\sum_i (Y_{i,\text{mediated}} - Y_{i,\text{baseline}})$.
$\hat{\Delta}_U$	Paired utility difference estimate	$\frac{1}{N}\sum_i (U_{i,\text{mediated}} - U_{i,\text{baseline}})$.
B	Number of resampling draws	The current fixture uses a configured deterministic resampling count.

Rates use the declared denominator of task cases in the relevant condition; trace-completeness rates use traces, and attack-success rates use declared attack cases. The current six-case record is a deterministic synthetic fixture. Its percentile-resampling intervals are descriptive summaries over paired cases, not population confidence intervals, p-values, or evidence of external effectiveness.

12.6 Cross-reference rule

Formal sections should cite this glossary when introducing a symbol, and captions should name a quantity in words when a reader could confuse it with a probability, posterior, or population estimate. The formalism inventory in Section 11 records which of these objects have executable translations and which remain partial or research-stage.

13 Appendix: Design Rationale, Assumptions, and Alternatives

This appendix records why the released Colony Kernel uses its present mechanisms and what each choice gives up. The decisions are engineering choices, not proofs that the selected policy is optimal. Where an alternative would require outcome data, the release treats calibration as future evaluation rather than retroactively presenting a hand-set constant as an empirical result.

The numeric values shown here are current implementation defaults or illustrative initial settings, not universal constants, fitted parameters, or empirical estimates. They are configurable and can be tuned or replaced through the corresponding runtime policy, configuration, or presentation inputs; after a change, regenerate the tests, tables, figures, and manuscript before interpreting the result. The appendix consequently distinguishes runtime defaults, presentation settings, and future-study inputs wherever their roles differ.

13.1 DR-1: Weighted additive gate with hard overrides

After its early-return checks, `ActuationGate` computes the score in Equation 25:

$$g = 0.3b + 0.3\rho(h) + 0.25u + 0.15c, \tag{25}$$

where b is budget credit, $\rho(h)$ is local hazard credit derived from the larger of RISK and FAILURE pressure, u is tiered trust credit, and c is proposal completeness. The score is a routing policy, not a probability of safety or harm.

Table 47 summarizes the relevant alternatives.

Table 47: Tradeoffs among candidate ordinary-score policies.

Policy family	Useful property	Cost or missing prerequisite
Weighted additive	Direct component decomposition; partial credit supports HOLD	Compensation among components; hand-set weights require external calibration
Multiplicative	A zero component can dominate without a separate rule	Small components suppress the whole score; interpretation depends on scaling
Explicit rule set	Named conditions are easy to audit	Interactions and ordering grow with the rule set
Learned scorer	Can estimate interactions from labeled outcomes	Requires representative, independently attested training and calibration data

The implementation does not rely on the weighted sum for every condition. Missing budget in the integrated path, SANDBOX status, trust below 0.3, and CRITICAL findings take explicit branches. This hybrid keeps the ordinary score inspectable while reserving non-compensatory treatment for named conditions. It also means that analysis of the formula alone is insufficient to predict every gate result.

13.2 DR-2: Subtractive, tick-driven signal expiry

Each trace stores an evaporation amount ϵ when deposited. A passive tick uses

$$x_{j,t+1} = \max(0.0, x_{j,t} - \epsilon_j), \tag{26}$$

and removes a trace at zero. At the defaults, FAST, NORMAL, and SLOW subtract 0.3, 0.1, and 0.02 per tick. Equation 26 is exact for a trace with no new deposit, reinforcement, or read-side effect.

The choice makes expiry finite and contract tests easy to replay. Its tradeoffs are equally explicit: ticks are logical rather than wall-clock time, a larger effective deposit lasts longer, and repeated deposits can dominate the nominal class. Source and trust multipliers affect initial strength, while the field cap bounds accumulated strength. Deployments that need real-time semantics must define the scheduler-to-tick mapping rather than reinterpret a tick as an undocumented number of seconds.

13.3 DR-3: Process-local state with optional file-backed consequences

`ColonyKernelConfig` defaults `db_path` to `:memory:`. In that mode consequence rows and profiles disappear when the process exits. `ConsequenceMemory` also supports a caller supplied SQLite file path, while `db_path=None` selects its pure in-memory list mode. The pheromone field remains in memory in all three cases.

This design minimizes setup and keeps real SQLite behavior available to tests. It does not provide shared multi-process state, restart recovery for signal pressure, remote replication, migration management, or conflict resolution. A deployment requiring those properties needs an explicit state architecture; changing only the consequence database path is not sufficient.

13.4 DR-4: Clipped additive trust updates

The integrated outcome path applies fixed deltas and clips trust to $[0.0, 1.0]$. A clean reported outcome adds 0.04; a failed one changes trust by -0.08; repair and human-feedback terms can change the net update. The rule is transparent and supports exact fixtures such as Table 24.

The cost of that simplicity is that trust is neither a calibrated posterior nor an uncertainty interval. Equal scores can arise from different histories, and constant-step updates do not imply convergence to a unique equilibrium. More importantly, ordinary MCP outcomes are caller supplied and do not consume a durable prior EXECUTE record. Optional and required kernel modes now support local proposal/verdict/authorization/ receipt/outcome linkage, with required mode rejecting the ordinary method. Before trust is used in an adversarial setting, that local integrity chain must also be bound to independently observed external execution and authenticated deployment identities.

13.5 DR-5: Three routing outcomes

EXECUTE, HOLD, and REFUSE distinguish three operational states:

- EXECUTE returns an advisory approval verdict to the caller;
- HOLD requests revision, evidence, budget recovery, or later reevaluation; and
- REFUSE rejects the proposal and, in the integrated path, deposits FAILURE pressure.

HOLD is useful only when revision can change a relevant input at acceptable cost. The release therefore does not claim that a three-way gate universally dominates a binary one. A benchmark must measure revision cost, delay, eventual decision, throughput, and outcome quality. The integrated and standalone budget paths also differ: an integrated budget failure yields HOLD, while the standalone gate with its own ledger returns REFUSE. Callers must choose the intended API rather than assume the labels are interchangeable.

13.6 DR-6: Deterministic falsification before scoring

The integrated proposal path runs deterministic checks before gate scoring. The checks cover 10 canonical `AttackVector` categories through 11 check functions. This order makes findings available to the gate and makes the same input replayable without an LLM call.

Deterministic heuristics have bounded, inspectable behavior, but they detect only the patterns they implement. The current checks top out at HIGH severity; CRITICAL remains an input class that would trigger the hard override. A clean report is not proof that a proposal is safe, and returning REFUSE does not physically prevent a separate caller from bypassing the kernel. Stronger deployments need an execution boundary that requires a consumed gate authorization, plus external isolation and monitoring.

13.7 DR-7: Role labels separated from action-type policy

The trust-score path maps profiles onto 5 labels: SANDBOX, REPAIR_ANT, MEMORY_ANT, DISPATCHER, and GUARD_ANT. The labels provide a readable lifecycle summary, and SANDBOX is an actual early gate refusal. The other four labels do not currently enforce distinct action-type matrices in `ActuationGate`.

This separation is important. A label taxonomy is inexpensive and auditable, but it is not a complete access-control system. The default lifecycle also has a bootstrap gap: new profiles begin in SANDBOX, the gate refuses their proposals, and trust rises only through submitted outcomes. An external study must either supply a fixed supervised calibration history or implement a narrowly constrained SANDBOX path with local lifecycle authentication and deployment-specific external observation.

13.8 DR-8: Shared location field rather than peer messaging

The pheromone field lets later proposals query pressure by location and signal type without reconstructing every prior consequence record. It supports the paper's bounded locality claim: a failure at one target can affect that target while an unrelated target remains unchanged.

The current implementation is a central in-process store, not a distributed stigmergic network. The release makes no asymptotic communication claim because actual cost depends on index structure, query pattern, agent topology, and synchronization design. A shared field reduces the API needed for the tested single-process case; it also creates a single-process state boundary and supplies no cross-host consistency protocol.

13.9 DR-9: Advisory pruning with a separate destructive API

`PruningDaemon.report()` and the MCP `colony_pruning_report` surface candidates; they do not move files. `PruningDaemon.archive(dry_run=True)` is also non-mutating by default. A direct caller can pass `dry_run=False`, in which case the implementation moves an existing in-repository path under `docs/plans/archived/` after a containment check.

Keeping discovery separate from mutation makes review possible and prevents the MCP report tool from silently archiving code. The tradeoff is procedural: safety depends on which callers can invoke the lower-level archive method and on reviewing false-positive candidates. Confidence scores are heuristic rankings, not calibrated probabilities, and DEPENDENCY pressure is a veto signal only within the implemented scan rules.

13.10 Generated figure accessibility and evidence inventory

The release route generates the 18 assets in Table 48. Captions in the body state the evidence class, fixed inputs, intended reading, and permitted inference. They are not reused as image alternatives: each figure instead has a concise structural alternative and a fuller description of layout, relationships, redundant encodings, and claim limits. This separation follows the distinction between chart semantics and prose description in accessible visualization research (Lundgard and Satyanarayan 2022), the evidence that visual communication depends on perceptual task and design rather than decoration (Franconeri et al. 2021), and guidance to provide a complete text equivalent for complex images (World Wide Web Consortium n.d.).

Colour remains useful for grouping, but it is not the only categorical channel. Labels, position, marker shape, line style, arrow direction, or printed status repeat the relevant distinction, and the palette is luminance-adjusted against the light figure background. This responds to documented scientific-communication failures caused by unsuitable colour maps (Crameri et al. 2020). The contract is intentionally bounded: metadata checks and contrast calculations do not prove usability with every reader, assistive technology, display, or print process.

Table 48: Generated figure evidence classes and text alternatives.

Asset and evidence class	Text alternative and extended description
`cover.png` — `schematic`	Short alternative: Codomyrmex cover: dark radial schematic of the Colony Kernel hub connected to 7 labelled control-plane subsystems. Extended description: A bright Colony Kernel hub sits at the centre of a dark field. Labelled nodes for pheromone storage, resources, gating, consequence memory, roles, pruning, and falsification surround it, joined by luminous spokes and signal trails. The composition is conceptual branding rather than a measured network.
`colony_pressure_loop.png` — `schematic`	Short alternative: Numbered circular flow from proposal submission through review, gate decision, caller actuation, outcome reporting, state update, and pheromone deposit. Extended description: Eight labelled cards form a clockwise loop. A proposal is falsification-checked, witnessed against state, and evaluated by the gate before a separate caller may actuate it. The caller then reports an outcome, which updates trust, budget, role, and local pheromone pressure for later proposals. Numbers and arrow direction, rather than colour alone, define the sequence.
`pheromone_decay.png` — `analytic`	Short alternative: Three directly labelled descending trajectories show FAST, NORMAL, and SLOW subtractive decay, each ending at its marked extinction tick. Extended description: All traces begin at strength 1.0 and decline linearly toward 0.0. FAST has the steepest slope and earliest vertical extinction marker, NORMAL is intermediate, and SLOW extends furthest to the right. Line style, endpoint label, and colour redundantly identify each class; the curves are analytical rather than observed time series.

Asset and evidence class	Text alternative and extended description
`gate_score_heatmap.png` — `analytic`	Short alternative: Heatmap of REFUSE, HOLD, and EXECUTE regions over trust and effective hazard, with decision names and thresholds printed inside the policy map. Extended description: Trust increases from left to right and effective local hazard increases from bottom to top. A dark left band marks the hard trust-floor refusal region. Beyond it, labelled REFUSE, HOLD, and EXECUTE regions change at the configured hazard and score boundaries. Region position, printed names, and colour all encode the decision; the grid is formula-derived, not an observed distribution.
`trust_trajectory.png` — `deterministic-fixture`	Short alternative: Stepwise trust trajectory crossing labelled role thresholds after clean reports, with the independent gate floor marked separately. Extended description: A monotone stair-step line starts at sandbox trust and rises after each caller-reported clean outcome. Horizontal threshold lines and directly named background bands show the inferred roles. A separate gate hard-floor line demonstrates that the first role promotion does not itself make ordinary scoring reachable. The path is a deterministic fixture, not a population trend.
`falsification_vectors.png` — `code-taxonomy`	Short alternative: Horizontal severity chart placing every falsification category in a labelled LOW, MEDIUM, HIGH, or CRITICAL band with an initial printed on each marker. Extended description: Each row names one live adversarial vector. A line ends at that vector's highest emitted severity rank, and the marker contains the severity initial. Rows are sorted from highest to lowest severity. The CRITICAL column is explicitly labelled even when empty because that class alone hard-refuses before scoring. The plot shows code taxonomy, not prevalence or detection rate.
`subsystem_architecture.png` — `schematic`	Short alternative: Hub-and-spoke diagram with ColonyKernel at the centre and 7 labelled subsystem nodes around it. Extended description: ColonyKernel occupies the central hub. Radial spokes connect it bidirectionally to the labelled Pheromone Store, Resource Ledger, Actuation Gate, Consequence Memory, Role Adapter, Pruning Daemon, and Falsification Worker nodes. Subtitles state each node's function, and arrowheads show information flow. The topology describes software ownership, not hosts, latency, or throughput.
`gate_score_3d.png` — `analytic`	Short alternative: Gate-score surface over trust and completeness with labelled HOLD and EXECUTE thresholds plus a projection of distinct trust slices. Extended description: The main panel rises across completeness and changes in discrete tiers across trust, with horizontal threshold planes marking HOLD and EXECUTE. A companion projection uses distinct line styles, point shapes, direct labels, and colour for selected trust slices. The continuous completeness surface is a visual envelope over a runtime input that is actually discrete.

Asset and evidence class	Text alternative and extended description
`fep_correspondence.png` — `conceptual-analogy`	Short alternative: Row-by-row correspondence diagram linking Free Energy Principle terms to named Colony Kernel artifacts and bounded engineering interpretations. Extended description: Each horizontal row begins with an FEP concept, points to a specific kernel artifact, and ends with an engineering interpretation. The repeated three-part layout and text labels carry the mapping; coloured row accents only aid scanning. A warning beneath the rows states that the current deterministic kernel is not a Bayesian controller and the correspondences are analogies, not equivalences.
`research_roadmap.png` — `research-plan`	Short alternative: 7 vertically connected milestone cards progress from implemented local contracts to a research-stage probabilistic extension. Extended description: Milestones run top to bottom through reproducibility, a local authenticated lifecycle ledger, external-actuation attestation, adversarial evaluation, calibration, persistence and concurrency, and a probabilistic extension. Every card prints its current status and an abbreviated required-evidence artifact. Directional arrows, sequence identifiers, and status text define the dependency order; colour is redundant and no milestone promises delivery or success.
`formalism_code_crosswalk.png` — `formal-crosswalk`	Short alternative: 7-row, three-column crosswalk linking each formal object and printed status to code translation, evidence, and a claim limit. Extended description: Each row identifies a formalism and explicitly prints IMPLEMENTED, PARTIAL, or RESEARCH. The middle column abbreviates its typed code anchors and translation bridge; the final column abbreviates executable evidence and the strongest claim the evidence does not support. Alternating row shading and status-colour strips aid navigation, while the full wording remains available in searchable tables.
`replay_contract.png` — `deterministic-fixture`	Short alternative: Three-stage paired-locality replay showing clear, failed-target, and recovered decisions while an unrelated target remains unchanged. Extended description: The first stage shows an EXECUTE decision for a clear target. After a caller-reported failure, the same target moves to HOLD while a paired unrelated target stays at EXECUTE. Passive decay then restores the original decision. Stage order, decision words, score labels, and arrows carry the comparison; colours are redundant and the fixture is not causal or externally attested.
`attestation_event_chain.png` — `authenticated-fixture`	Short alternative: Top-to-bottom authenticated event chain from proposal through verdict, authorization, execution receipt, and outcome. Extended description: Five numbered event nodes are connected top to bottom by directional arrows. Each row prints its event type, actor, hash prefix, and signature state. The chain order binds proposal, gate verdict, authorization, caller-supplied execution receipt, and caller-supplied outcome digests under the configured signer. The linked structure establishes fixture-level ledger integrity only; it does not verify external actuation, safety, or usefulness.

Asset and evidence class	Text alternative and extended description
`safety_utility_frontier.png` — `offline-synthetic`	Short alternative: Scatter plot comparing directly labelled always-execute and independent-reference-gate conditions on harmful-action rate and fixture utility, using different marker shapes. Extended description: Harmful-action rate is on the horizontal axis and utility is on the vertical axis, both averaged over the same ordered synthetic task cases. One labelled point represents the always-execute baseline and a differently shaped labelled point represents the independent ReferenceGate-mediated condition; an arrow annotates the paired mediated-minus-baseline harm difference and its descriptive interval. The points come from deterministic synthetic cases. The mediator is not the production ActuationGate, parity is not established, and the interval does not imply population generalization or calibration.
`calibration_reliability.png` — `calibration-status`	Short alternative: Reliability-diagram placeholder with an ideal diagonal and a prominent not_estimated status notice instead of invented calibration data. Extended description: The square plotting area shows only the ideal-reference diagonal. A central status card states that calibration is not_estimated and that expected calibration error is not_run. No bins or empirical points are drawn because gate and trust scores are not probabilities and the required held-out outcomes and declared confidence values are absent.
`persistence_recovery.png` — `restart-fixture`	Short alternative: Restart-fixture bar chart comparing signal strength at deposit, after reopening SQLite storage, and after recovery, with values printed above the bars. Extended description: Three ordered bars report the signal's strength when deposited, after the store is closed and reopened, and at the fixture's recovery observation. Numeric labels are printed above the bars so height and colour are not the only encodings. The retained record supports a narrow logical durability check, not throughput, arbitrary crash survival, or persistence of all kernel state.
`formalism_coverage.png` — `formalism-inventory`	Short alternative: Stacked status summary of the formalism inventory with implemented, partial, and research counts printed in their respective segments. Extended description: One horizontal bar partitions all 7 configured formalism mappings by current bridge status. Each non-empty segment prints its status and count, while a legend repeats the categories. Segment length, order, text, and colour provide redundant encodings. The inventory counts documented mappings, not proved theorems or semantic equivalences.
`research_status_matrix.png` — `research-status`	Short alternative: Status dot matrix listing roadmap milestones and formalism mappings by labelled status columns, with one marker and row label per item. Extended description: Rows name configured roadmap stages and formalism mappings. Each row places a single marker under its printed IMPLEMENTED, PARTIAL, NEXT, PLANNED, or RESEARCH column, so horizontal position and text define the status independently of colour. The matrix is a configuration-backed navigation aid, not evidence that conditional work is complete or empirically successful.

Every generator reads the manuscript variable snapshot and stamps a version, compact configuration digest, and generation date. As explained in Section 8, that footer identifies the manuscript configuration used by the figure route; it does not hash all source or authenticate the image. The generated `figure_registry.json` adds the caption, concise alternative, extended description, evidence-class label, byte size, and full SHA-256 for each PNG file. It is an accessibility and integrity inventory for the emitted

files, not an external signature or a usability study.

13.11 Calibration and replacement criteria

The gate weights and thresholds were selected as design constants. Replacing them with new hand-set values would change policy, not add evidence. A defensible calibration study should provide representative proposals, consumed execution records, independently observed and externally attested outcomes, costs of HOLD/revision, pre-registered metrics, held-out evaluation, and calibration diagnostics. A learned policy should remain subordinate to named hard conditions unless the study separately justifies changing those conditions.

The same replacement discipline applies to every decision in this appendix. A real-time field should be judged against replayability and scheduler failures; a durable state backend against migration and recovery tests; role-specific action policies against an explicit matrix; and automated pruning against measured false-positive costs. The present design is valuable because its boundaries are testable—not because it is the final design for every deployment.

13.12 Summary

The released design favors small deterministic mechanisms with inspectable inputs and explicit limitations. That choice enables the paired locality result and exact policy figures reported in Section 5. It does not establish external effectiveness, adversarial integrity, restart continuity, or optimal calibration. Those claims require the additional evidence package specified in Table 36.

Acknowledgements

We thank Marek Pawel Bargiel for conceptual comments on pressure-aware gating / colony-control framing.

References

AI, LangChain. 2024. *LangGraph: Build Stateful, Multi-Actor Applications with LLMs*. https://github.com/langchain-ai/langgraph.

Alshiekh, Mohammed, Roderick Bloem, Ruediger Ehlers, Bettina Konighofer, Scott Niekum, and Ufuk Topcu. 2018. "Safe Reinforcement Learning via Shielding." *Proceedings of the AAAI Conference on Artificial Intelligence* 32. https://doi.org/10.1609/aaai.v32i1.11797.

Apt, Krzysztof R. 2003. *Principles of Constraint Programming*. Cambridge University Press.

Bonabeau, Eric, Marco Dorigo, and Guy Theraulaz. 1999. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press.

Buhl, Marie Davidsen, Gaurav Sett, Leonie Koessler, Jonas Schuett, and Markus Anderljung. 2024. "Safety Cases for Frontier AI." *arXiv Preprint arXiv:2410.21572*. https://arxiv.org/abs/2410.21572.

Crameri, Fabio, Grace E. Shephard, and Philip J. Heron. 2020. "The Misuse of Colour in Science Communication." *Nature Communications* 11: 5444. https://doi.org/10.1038/s41467-020-19160-7.

Debenedetti, Edoardo, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramer. 2024. "AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents." *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*. https://arxiv.org/abs/2406.13352.

Dorigo, Marco, and Thomas Stützle. 2004. *Ant Colony Optimization*. MIT Press.

Dwork, Cynthia, and Aaron Roth. 2014. "The Algorithmic Foundations of Differential Privacy." *Foundations and Trends in Theoretical Computer Science* 9 (3–4): 211–407. https://doi.org/10.1561/0400000042.

Franconeri, Steven L., Lace M. Padilla, Priti Shah, Jeffrey M. Zacks, and Jessica Hullman. 2021. “The Science of Visual Data Communication: What Works.” *Psychological Science in the Public Interest* 22 (3): 110–61. https://doi.org/10.1177/15291006211051956.

Friston, Karl. 2010. “The Free-Energy Principle: A Unified Brain Theory?” *Nature Reviews Neuroscience* 11 (2): 127–38. https://doi.org/10.1038/nrn2787.

Friston, Karl, Thomas FitzGerald, Francesco Rigoli, Philipp Schwartenbeck, and Giovanni Pezzulo. 2017. “Active Inference: A Process Theory.” *Neural Computation* 29 (1): 1–49. https://doi.org/10.1162/NECO_a_00912.

Grassé, Pierre-Paul. 1959. “La Reconstruction Du Nid Et Les Coordinations Inter-Individuelles Chez Bellicositermes natalensis Et Cubitermes Sp. La Théorie de La Stigmergie: Essai d’interprétation Du Comportement Des Termites Constructeurs.” *Insectes Sociaux* 6 (1): 41–80. https://doi.org/10.1007/BF02223791.

Greshake, Kai, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. 2023. “Not What You’ve Signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection.” *arXiv Preprint arXiv:2302.12173*. https://arxiv.org/abs/2302.12173.

Huynh, Trung Dong, Nicholas R. Jennings, and Nigel Shadbolt. 2006. “An Integrated Trust and Reputation Model for Open Multi-Agent Systems.” *Autonomous Agents and Multi-Agent Systems* 13 (2): 119–54. https://doi.org/10.1007/s10458-005-6825-4.

Inc., CrewAI. 2024. *CrewAI: Framework for Orchestrating Role-Playing, Autonomous AI Agents*. https://github.com/crewAIInc/crewAI.

Kamvar, Sepandar D., Mario T. Schlosser, and Hector Garcia-Molina. 2003. “The EigenTrust Algorithm for Reputation Management in P2P Networks.” *Proceedings of the 12th International Conference on World Wide Web*, 640–51. https://doi.org/10.1145/775152.775242.

Karpas, Ehud, Omri Abend, Yonatan Belinkov, et al. 2022. “MRKL Systems: A Modular, Neuro-Symbolic Architecture That Combines Large Language Models, External Knowledge Sources and Discrete Reasoning.” *arXiv Preprint arXiv:2205.00445*. https://arxiv.org/abs/2205.00445.

Liu, Xiao, Hao Yu, Hanchen Zhang, et al. 2023. “AgentBench: Evaluating LLMs as Agents.” *arXiv Preprint arXiv:2308.03688*. https://arxiv.org/abs/2308.03688.

Lundgard, Alan, and Arvind Satyanarayan. 2022. “Accessible Visualization via Natural Language Descriptions: A Four-Level Model of Semantic Content.” *IEEE Transactions on Visualization and Computer Graphics* 28 (1): 1073–83. https://doi.org/10.1109/TVCG.2021.3114770.

Marsh, Stephen Paul. 1994. “Formalising Trust as a Computational Concept.” PhD thesis, University of Stirling. https://dspace.stir.ac.uk/handle/1893/2010.

Miller, Mark S., and Jonathan S. Shapiro. 2003. “Paradigm Regained: Abstraction Mechanisms for Access Control.” *Advances in Computing Science—ASIAN 2003*, 224–42. https://doi.org/10.1007/978-3-540-40965-6_15.

Mitchell, Margaret, Simone Wu, Andrew Zaldivar, et al. 2019. “Model Cards for Model Reporting.” *Proceedings of the Conference on Fairness, Accountability, and Transparency*, 220–29. https://doi.org/10.1145/3287560.3287596.

Model Context Protocol Contributors. 2026. *Model Context Protocol Specification*. https://modelcontextprotocol.io/specification/2026-07-28.

Nakano, Reiichiro, Jacob Hilton, Suchir Balaji, et al. 2021. “WebGPT: Browser-Assisted Question-Answering with Human Feedback.” *arXiv Preprint arXiv:2112.09332*. https://arxiv.org/abs/2112.09332.

Parunak, H. Van Dyke. 1997. “”Go to the Ant”: Engineering Principles from Natural Multi-Agent Systems.” *Annals of Operations Research* 75: 69–101. https://doi.org/10.1023/A:1018980001403.

Patil, Shishir G., Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. “Gorilla: Large Language Model Connected with Massive APIs.” *arXiv Preprint arXiv:2305.15334*. https://arxiv.org/abs/2305.15334.

Pearl, Judea. 1988. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann.

Peng, Roger D. 2011. "Reproducible Research in Computational Science." *Science* 334 (6060): 1226–27. https://doi.org/10.1126/science.1213847.

Popper, Karl R. 2002. *The Logic of Scientific Discovery*. 2nd ed. Routledge.

Qin, Yujia, Shihao Liang, Yining Ye, et al. 2023. "ToolLLM: Facilitating Large Language Models to Master 16000+ Real-World APIs." *arXiv Preprint arXiv:2307.16789*. https://arxiv.org/abs/2307.16789.

Raji, Inioluwa Deborah, Andrew Smart, Rebecca N. White, et al. 2020. "Closing the AI Accountability Gap: Defining an End-to-End Framework for Internal Algorithmic Auditing." *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 33–44. https://doi.org/10.1145/3351095.3372873.

Ramstead, Maxwell J. D., Axel Constant, Paul B. Badcock, and Karl J. Friston. 2019. "Variational Ecology and the Physics of Sentient Systems." *Physics of Life Reviews* 31: 188–205. https://doi.org/10.1016/j.plrev.2018.12.002.

Rose, Scott, Oliver Borchert, Stu Mitchell, and Sean Connelly. 2020. *Zero Trust Architecture*. NIST Special Publication Nos. 800-207. National Institute of Standards; Technology. https://doi.org/10.6028/NIST.SP.800-207.

Ruan, Yangjun, Honghua Dong, Andrew Wang, et al. 2023. "Identifying the Risks of LM Agents with an LM-Emulated Sandbox." *arXiv Preprint arXiv:2309.15817*. https://arxiv.org/abs/2309.15817.

Sabater, Jordi, and Carles Sierra. 2005. "Review on Computational Trust and Reputation Models." *Artificial Intelligence Review* 24 (1): 33–60. https://doi.org/10.1007/s10462-004-0041-5.

Saltzer, Jerome H., and Michael D. Schroeder. 1975. "The Protection of Information in Computer Systems." *Proceedings of the IEEE* 63 (9): 1278–308. https://doi.org/10.1109/PROC.1975.9939.

Schick, Timo, Jane Dwivedi-Yu, Roberto Dessì, et al. 2023. "Toolformer: Language Models Can Teach Themselves to Use Tools." *arXiv Preprint arXiv:2302.04761*. https://arxiv.org/abs/2302.04761.

Seto, David, Bruce Krogh, Lui Sha, and Alongkrit Chutinan. 1998. "The Simplex Architecture for Safe Online Control System Upgrades." *Proceedings of the 1998 American Control Conference*, 3504–8. https://doi.org/10.1109/ACC.1998.703255.

Trivedi, Harsh, Tushar Khot, Mareike Hartmann, et al. 2024. "AppWorld: A Controllable World of Apps and People for Benchmarking Interactive Coding Agents." *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*. https://arxiv.org/abs/2407.18901.

Wooldridge, Michael, and Nicholas R. Jennings. 1995. "Intelligent Agents: Theory and Practice." *The Knowledge Engineering Review* 10 (2): 115–52. https://doi.org/10.1017/S0269888900008122.

World Wide Web Consortium. n.d. *Images Tutorial*. Web Accessibility Initiative. https://www.w3.org/WAI/tutorials/images/.

Wu, Qingyun, Gagan Bansal, Jieyu Zhang, et al. 2023. "AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation." *arXiv Preprint arXiv:2308.08155*. https://arxiv.org/abs/2308.08155.

Xie, Tianbao, Danyang Zhang, Jixuan Chen, et al. 2024. "OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments." *arXiv Preprint arXiv:2404.07972*. https://arxiv.org/abs/2404.07972.

Yang, John, Carlos E. Jimenez, Alexander Wettig, et al. 2024a. "SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering." *Advances in Neural Information Processing Systems*. https://arxiv.org/abs/2405.15793.

Yang, John, Carlos E. Jimenez, Alexander Wettig, et al. 2024b. "SWE-bench: Can Language Models Resolve Real-World GitHub Issues?" *arXiv Preprint arXiv:2310.06770*. https://arxiv.org/abs/2310.06770.

Yao, Shunyu, Jeffrey Zhao, Dian Yu, et al. 2023. "ReAct: Synergizing Reasoning and Acting in Language Models." *International Conference on Learning Representations*. https://arxiv.org/abs/2210.03629.

Zhan, Qiusi, Zhixiang Liang, Zifan Ying, and Daniel Kang. 2024. “InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents.” *arXiv Preprint arXiv:2403.02691*. https://arxiv.org/abs/2403.02691.

Zhang, Zhexin, Shiyao Cui, Yida Lu, et al. 2024. “Agent-SafetyBench: Evaluating the Safety of LLM Agents.” *arXiv Preprint arXiv:2412.14470*. https://arxiv.org/abs/2412.14470.

Zhou, Shuyan, Frank F. Xu, Hao Zhu, et al. 2023. “WebArena: A Realistic Web Environment for Building Autonomous Agents.” *arXiv Preprint arXiv:2307.13854*. https://arxiv.org/abs/2307.13854.