Cortex: A Fixed-Point Theory of Governed Coding Agents

An empirical study of safety and capability under a governance layer

Marius-Constantin Dinu
Alpha Omega Labs

Florian Zeba
Alpha Omega Labs

Cortex Research · Technical report · June 2026

Abstract

A coding agent is a large language model (LLM) wrapped in a control loop – a harness – that lets it plan, write, and execute code. Raw harnesses optimize next-token capability, not governed behavior: under adversarial input they can be induced to exfiltrate secrets or run destructive commands, and over a long task they silently abandon requirements. We study Cortex, a meta-level control layer that supervises a base harness. Cortex couples three faculties over the base agent: governance – deterministic pre-execution checks and capability/dependency policies that constrain which actions may run; orchestration – instruction analysis and long-horizon planning that decompose a task into a tracked requirement structure with milestones and a semantic contract; and a validate–repair loop that drives those requirements to verified completion. In control-theoretic terms it is a supervisory controller, and in AI terms a metareasoner over the base policy: it decides not only whether an action is admissible but what the agent should attempt next and whether further computation is worthwhile – direction a guardrail layer alone cannot provide.

Our contributions are fourfold. First, we give Cortex a precise semantics: its governance checks are a projection onto an admissible action set, and its planning–validate–repair loop is an inflationary, monotone operator on the lattice of satisfied requirements. Second, we prove that this loop converges to a least fixed point in a bounded number of productive iterations, is sound with respect to a validation oracle, and is independent of execution order (Theorem 1). Third, we define chance-calibrated evaluation metrics for safety and capability and characterize their range and calibration. Fourth, as a non-replacing addition, we add a probabilistic execution view: a monotone Markov transition layer over the same requirement lattice. This layer does not define correctness – the fixed-point semantics do that – but it makes iteration count, reasoning budget, completion probability, expected hitting time, and risk reduction measurable. Empirically, across five task families, placing the same base model behind Cortex sharply reduces adversarial attack-success while preserving or improving capability, with the largest gains over long horizons where single-pass harnesses decay.

Contributions.

- A formal model of an LLM coding agent under a meta-level control layer (Sections 3–4).
- A convergence, soundness, and confluence theorem for the completion loop, grounded in the least-fixed-point semantics of monotone operators (Theorem 1).

- A bridge from this semantics to effective-feedback scaling and its decision-theoretic basis – value of information and value of computation (Section 4.3, Proposition 1).
- Closed-form, chance-calibrated evaluation metrics with proven range and monotonicity, plus a contamination-controlled protocol for the baseline comparison (Sections 5–6).
- A minimal probabilistic execution layer, kept separate from the correctness layer, that lets the same result be reported as completion probability, expected number of iterations, hook-induced risk reduction, and information in bits (Sections 4.4 and 5.7).

1 Introduction

An LLM coding harness manages context and turns model output into executed actions – shell commands, file edits, and tool calls. Capability and control are distinct axes: a highly capable harness can still be talked into an unsafe action, and a safe but myopic one can drop requirements partway through a long task. Cortex is a meta-level control layer that supplies what the base harness lacks on each axis. For safety it adds governance: deterministic pre-execution checks and capability/dependency policies that constrain which actions may run. For capability it adds orchestration: instruction analysis and long-horizon planning that structure the task. Tying the two together is an iterative validate–repair loop – the Synapse loop – that drives the task to verified completion. Cortex does not replace the base model; it supervises it.

The distinction matters. Governance alone – deterministic filtering of individual actions – can neither plan a long-horizon task nor judge whether a language-level request is safe to act on; a meta-level controller both constrains and directs, and its validation may be semantic (an LLM-checked requirement or contract) as well as a fixed rule. The empirical comparisons in this report come from a reproducible benchmark we refer to as Gauntlet.

Test-time scaling improves agents by spending more inference compute – repeated sampling, search, and revision – extending the training scaling laws to inference. Recent evidence indicates that the predictive quantity is not raw compute: effective feedback compute (EFC) credits feedback only when it is informative, valid, non-redundant, and retained. A control layer raises the quality of feedback: its checks discard invalid actions, its validation oracle admits only feedback that changes the task state, and its append-only ledger retains that feedback so it is never re-derived; its coordinator decides to iterate or stop, allocating compute by its value.

We address two questions: (i) what does a meta-level control layer guarantee? and (ii) how much does it improve a base model? We answer (i) by modeling governance checks as a projection onto an admissible action set and the Synapse loop as a monotone consequence operator whose least fixed point is the completed task, yielding convergence and soundness. We answer (ii) with closed-form, calibrated metrics and a contamination-controlled comparison in which baselines run with the layer removed. The probabilistic additions in this revision answer a third, operational question: given a fixed base policy, hook set, and reasoning budget, how likely and how fast is completion?

2 Related Work and Theoretical Lineage

Neurosymbolic and cognitive computation. A neural generator constrained and verified by a symbolic layer is the defining shape of neurosymbolic AI. The completion loop is, structurally, the

recognize–act cycle of classical cognitive architectures – the problem-space hypothesis of Newell and Simon, SOAR, and ACT-R – in which rules fire repeatedly until a goal or impasse. We make that cycle precise as a fixed-point computation. The framework formalized here is an instance of logic-and-generative-model integration.

Fixed-point semantics. Our convergence argument rests on least-fixed-point semantics for monotone operators: the immediate-consequence operator of logic programming, the Knaster–Tarski fixed-point theorem, and Kleene iteration. Where an operator is a contraction on a metric space, Banach's theorem gives uniqueness and a rate; we use the lattice formulation because the requirement state space is naturally a finite complete lattice.

Evaluation. For capability over trajectories we adopt VERTEX (defined in Section 5.1), whose presence component is a BERTScore-style bidirectional matching and whose order component is a distance-decayed dynamic time warping. This continues a line of work that scores agent trajectories for capability and quality rather than only final outputs. Code capability uses the unbiased pass@k estimator; success rates carry Wilson score intervals and bootstrap intervals. Qualitative judging follows the LLM-as-judge literature, with position/verbosity-bias controls; safety draws on jailbreak and real-world prompt-injection literatures.

Scaling, test-time compute, effective feedback, and stochastic execution. Training scaling laws relate compute, data, and loss by power laws; test-time scaling extends this to inference, where repeated sampling, search, and revision trade compute for accuracy. Effective feedback compute argues that the predictive coordinate for agent-harness scaling is not raw expenditure but feedback that is informative, valid, non-redundant, and retained. The fixed-point loop gives this coordinate a deterministic semantics. The Markov layer added in Section 4.4 gives it an operational probability semantics: it does not change what counts as correct, but it quantifies completion probability, expected hitting time, and the effect of hooks and reasoning budgets on the distribution over final states.

Iterative feedback, verification, and metareasoning. Closed-loop harnesses that reason, act, and revise – ReAct, Self-Refine, Reflexion – and step-level verification/process rewards are heuristic instances of execute–validate–repair. They typically lack a convergence or soundness guarantee; we supply one by casting the loop as a monotone consequence operator. The decision-theoretic basis for which feedback to credit and when to keep computing is classical: value of information assigns positive worth only to information that can change a decision, and rational metareasoning selects computations by their value under bounded resources.

3 Preliminaries and Notation

We write $\mathbb{N}$ for the naturals, $\mathbb{R}$ for the reals, and $\mathbb{I} := [0, 1]$ for the unit interval. For a finite set X, $|X|$ is its cardinality and 2^X its power set. The probability simplex over X is

$$\Delta(X) := \left\{ p \in \mathbb{R}_{\geq 0}^{X} : \sum_{x \in X} p_x = 1 \right\}.$$

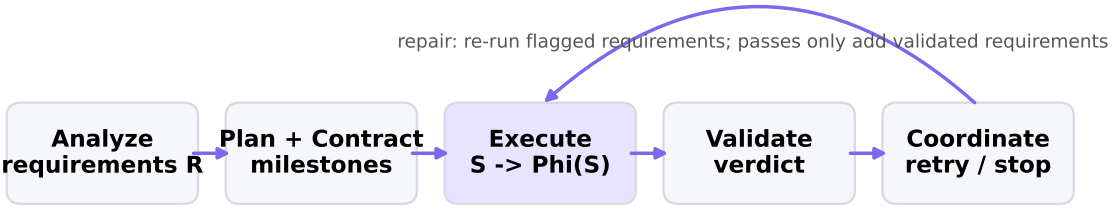

Figure 1: The Cortex completion (Synapse) loop. A task is analyzed into a requirement set R and a plan/contract; the agent repeatedly executes against open requirements, each pass applies the consequence operator Φ, and the coordinator either repairs flagged requirements or stops when a pass adds nothing, i.e. at a fixed point $\Phi(S) = S$.

We write $S^{d-1} := \{v \in \mathbb{R}^d : \|v\|_2 = 1\}$ for the unit sphere in $\mathbb{R}^d$, with Euclidean inner product $\langle \cdot, \cdot \rangle$, so $\langle u, v \rangle \in [-1, 1]$ for $u, v \in S^{d-1}$. We write

$$\mathrm{clamp}(x; a, b) := \min(\max(x, a), b)$$

for clamping a real number to $[a, b]$.

Order-theoretic objects. A complete lattice is a partially ordered set $(L, \sqsubseteq)$ in which every subset has a least upper bound and greatest lower bound; it has a bottom $\bot$ and top $\top$. A map $f : L \to L$ is monotone if $x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$, and inflationary if $x \sqsubseteq f(x)$ for all x. A fixed point satisfies $f(x) = x$; by Knaster–Tarski a monotone map on a complete lattice has a least fixed point $\mathrm{lfp}(f)$.

Agent objects. Let $\mathcal{A}$ be the countable action space of candidate actions a harness may propose, and $\mathcal{H}$ the history space of finite sequences of action–observation pairs. A base harness is a policy

$$\pi : \mathcal{H} \to \Delta(\mathcal{A}).$$

A hook is a predicate $h : \mathcal{A} \to \{0, 1\}$, where 1 means block.

Task objects. A task induces a finite requirement set

$$R = \{r_1, \ldots, r_N\}, \qquad N = |R| < \infty,$$

with a distinguished required subset $R_{\mathrm{req}} \subseteq R$. The state is the subset $S \subseteq R$ of requirements already satisfied with evidence; the state space is the finite complete lattice $(2^R, \subseteq)$, with join $\cup$, meet $\cap$, bottom $\emptyset$, and top R. A validation oracle is a map

$$\nu : 2^R \times R \to \{0, 1\},$$

where $\nu(S, r) = 1$ iff the evidence accumulated in S satisfies requirement r.

Metric objects. Let $\mathcal{D}$ be the space of finite behavior descriptors (natural-language strings). A trajectory is a finite sequence $(x_1, \ldots, x_m) \in \mathcal{D}^m$. A sentence embedding is a map $\varphi : \mathcal{D} \to S^{d-1}$. Quality signals take values $s \in \mathbb{I}$. Over K such signals, the weights form a convex vector $w \in \Delta(\{1, \ldots, K\})$, so $w_k \geq 0$ and $\sum_{k=1}^{K} w_k = 1$.

4 The Control Layer: Governance and Orchestration

Cortex supervises the base agent along two formal faculties: governance constrains admissible actions (Section 4.1), and orchestration – planning plus an iterative validate–repair loop – directs the agent to verified completion (Section 4.2). Section 4.3 relates the loop to effective-feedback scaling. Section 4.4 adds a non-replacing probabilistic execution layer.

4.1 Governance as a projection on the action space

Let $h_1, \ldots, h_m : \mathcal{A} \to \{0, 1\}$ be the m safety hooks, with $h_j(a) = 1$ blocking action a. Define the admissible set

$$\mathcal{A}_G := \left\{ a \in \mathcal{A} : \sum_{j=1}^{m} h_j(a) = 0 \right\}$$

and the governance map $G : \mathcal{A} \to \mathcal{A} \cup \{\bot\}$:

$$G(a) = \begin{cases} a, & a \in \mathcal{A}_G, \\ \bot, & \text{otherwise.} \end{cases}$$

The governed policy is: sample $a \sim \pi(\cdot \mid H)$, emit $G(a)$. Since each h_j is a pure predicate evaluated before execution, G restricted to $\mathcal{A}_G$ is the identity, so $G \circ G = G$ on $\mathcal{A}_G$: governance is a projection, and no action in $\mathcal{A} \setminus \mathcal{A}_G$ is ever executed. The safety hooks shrink the reachable action set without altering the base model's competence on admissible actions.

Assumption 1 (Sound, conservative hooks)**.** *Each h_j has no false negatives on the unsafe class it defines and is conservative elsewhere, so its false-positive rate is bounded by the rule's specificity.*

4.2 Orchestration: the completion loop as a consequence operator

One pass of the loop – execute against the open requirements, observe, validate via ν – induces an operator

$$\Phi : 2^R \to 2^R, \qquad \Phi(S) = S \cup \{r \in R : \nu(S, r) = 1\}.$$

The progress ledger is append-only: a requirement satisfied with evidence is never retracted. Hence Φ is inflationary,

$$S \subseteq \Phi(S),$$

and, because additional evidence never invalidates a satisfied requirement, monotone,

$$S \subseteq T \Rightarrow \Phi(S) \subseteq \Phi(T).$$

Theorem 1 (Convergence, soundness, confluence)**.** *Let $\Phi : 2^R \to 2^R$ be inflationary and monotone on the finite complete lattice $(2^R, \subseteq)$. Then:*

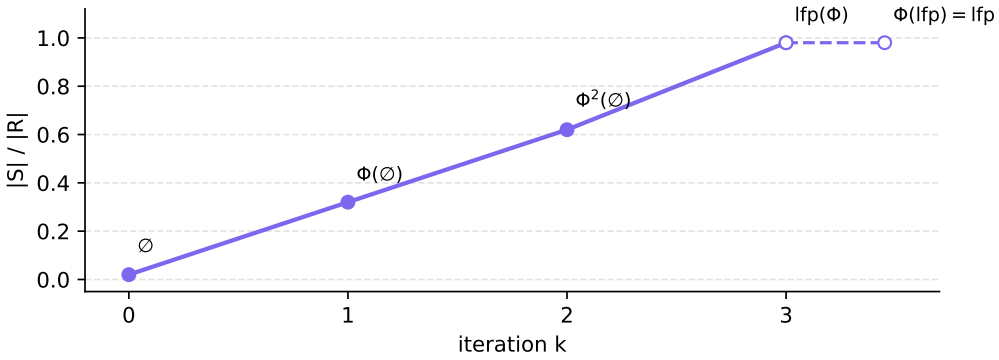

Figure 2: Convergence of the loop, drawn as a Kleene ascent. The horizontal axis is iteration k; the vertical axis is the fraction of requirements satisfied. Because the append-only ledger makes the iterates a non-decreasing chain bounded above by $|R|$, the chain reaches the least fixed point in at most $|R|$ productive steps.

(a) the Kleene iterates

$$\emptyset \subseteq \Phi(\emptyset) \subseteq \Phi^2(\emptyset) \subseteq \cdots$$

form a non-decreasing chain that stabilizes at the least fixed point $\mathrm{lfp}(\Phi)$ *after at most* $|R|$ *productive iterations;*

(b) the loop returns PASS iff $R_{\mathrm{req}} \subseteq \mathrm{lfp}(\Phi)$*;*

(c) $\mathrm{lfp}(\Phi)$ *is independent of the order in which consequences are added.*

Proof. Existence of $\mathrm{lfp}(\Phi)$ is Knaster–Tarski. For (a), define the ranking function $\psi : 2^R \to \mathbb{N}$ by

$$\psi(S) = |R \setminus S|.$$

On any productive pass, $\Phi(S) \supsetneq S$, so ψ strictly decreases; since $\psi \geq 0$ is bounded below, the descent is well-founded and there are at most $|R|$ productive passes before a fixed point $\Phi(S) = S$ is reached. Statement (b) is the definition of PASS together with soundness of ν: a requirement enters $\mathrm{lfp}(\Phi)$ only when ν certifies it, so $\mathrm{lfp}(\Phi)$ is exactly the set of achievable validated requirements. For (c), monotone operators on a complete lattice have a least fixed point that is the supremum of the Kleene chain, independent of the schedule by which consequences are added. □

A raw harness performs a single application of an ungoverned executor and stops – at best it computes $\Phi(\emptyset)$, with no guarantee that R_{req} is covered. The completion loop computes the closure

$$\Phi^*(\emptyset) = \mathrm{lfp}(\Phi),$$

and the stop condition, halt when a pass adds nothing, is exactly fixed-point detection.

4.3 Effective feedback and the value of computation

Recent evidence indicates that agent-harness performance scales with effective feedback compute (EFC) rather than raw compute. The completion loop realizes the criteria exactly. Call an iteration $S \mapsto \Phi(S)$ productive if $\Phi(S) \supsetneq S$, i.e. it adds at least one requirement. A productive iteration adds some r with $\nu(S, r) = 1$, which is informative because it changes the state, valid because it is

admitted only on the oracle's certificate, non-redundant because $r \notin S$, and retained because the ledger is append-only. We therefore define the effective feedback delivered by a pass as

$$\mathrm{efc}(S) = |\Phi(S) \setminus S|,$$

and the loop's coordinator as a metareasoner that continues while the value of the next computation is positive, $\mathrm{efc} > 0$, and stops at the fixed point where it is zero.

Proposition 1 (Effective-feedback accounting)**.** *Along the Kleene run $\emptyset, \Phi(\emptyset), \ldots, \mathrm{lfp}(\Phi)$, the total effective feedback equals the size of the fixed point:*

$$\sum_t \mathrm{efc}\big(\Phi^t(\emptyset)\big) = |\,\mathrm{lfp}(\Phi)|.$$

Under task-sufficiency, $R_{\mathrm{req}} \subseteq \mathrm{lfp}(\Phi)$, the loop succeeds after delivering the required feedback units. The task's required feedback demand is

$$D_{\mathrm{task}} := |R_{\mathrm{req}}|.$$

Proof. The iterates form a strictly increasing chain on productive passes, so the disjoint increments $\Phi^{t+1} \setminus \Phi^t$ partition $\mathrm{lfp}(\Phi)$. Summing their sizes telescopes to $|\,\mathrm{lfp}(\Phi)|$. Each productive pass contributes at least one unit, giving at most $|R|$ productive passes. Task-sufficiency makes R_{req} a subset of the partitioned set, so its requirements are each credited exactly once. □

Wording correction. A productive pass contributes one or more units of validated, retained feedback, not necessarily exactly one. The number of units is precisely $\mathrm{efc}(S) = |\Phi(S) \setminus S|$. Thus the iteration bound is at most $|R|$, while the number of productive passes can be smaller when one pass validates multiple requirements.

Two consequences follow. First, the loop spends no compute re-deriving retained feedback, so its conversion of raw passes into effective feedback is limited only by the base model's per-pass productivity and the oracle's admissibility. Second, the bound of Theorem 1 is a semantic upper bound, while empirical reporting should also include the observed number of loop passes, productive passes, and requirements added per pass.

4.4 Probabilistic execution layer over the fixed-point lattice

The fixed-point semantics define correctness: what may enter the ledger and when the loop has completed. They do not by themselves describe how likely a stochastic agent is to reach completion within a budget. For that operational question, we add a Markov transition layer over the same requirement lattice. This is an addition, not a replacement.

Let

$$L = 2^R$$

be the finite requirement lattice. A Markov transition matrix over requirement states is

$$P : 2^R \times 2^R \to [0,1], \qquad \sum_{S' \subseteq R} P(S, S') = 1.$$

To preserve the monotone ledger semantics, impose

$$P(S, S') = 0 \quad \text{unless} \quad S \subseteq S'.$$

Thus a pass may stay at S or move upward to a superset, but it may not remove validated requirements.

The paper already defines a stochastic policy $\pi : \mathcal{H} \to \Delta(\mathcal{A})$. If actions and observations are sampled as

$$a_t \sim \pi_G(\cdot \mid H_t), \qquad o_t \sim E(\cdot \mid H_t, a_t),$$

a requirement update can be written as

$$S_{t+1} = U(S_t, a_t, o_t) = S_t \cup \{r \in R : \nu(S_t, a_t, o_t, r) = 1\}.$$

When the requirement state is a sufficient summary, the induced transition is

$$P(S, S') = \sum_{a,o:\, U(S,a,o)=S'} \pi_G(a \mid S) E(o \mid S, a).$$

Strictly, because the base policy is history-dependent, the genuinely Markovian state is

$$X_t = (H_t, S_t),$$

with transition kernel

$$K\big((H_{t+1}, S_{t+1}) \mid (H_t, S_t)\big).$$

The requirement-only matrix P is therefore a reporting abstraction or marginalization of the history-state process.

Hook-aware transitions. Governance can be incorporated by filtering the policy. For a hook predicate h, define

$$\pi_h(a \mid H) = \frac{\pi(a \mid H)\mathbf{1}\{h(a) = 0\}}{\sum_{b \in \mathcal{A}} \pi(b \mid H)\mathbf{1}\{h(b) = 0\}}.$$

Blocked actions receive probability zero and the remaining admissible actions are renormalized. Comparing P with the hook-constrained kernel P_h quantifies the safety effect of governance: how much probability mass is shifted away from unsafe, incomplete, or delayed terminal states.

Completion as hitting time. Define the successful absorbing set

$$C = \{S \subseteq R : R_{\text{req}} \subseteq S\}.$$

The hitting time is

$$T_C = \inf\{t \geq 0 : S_t \in C\}.$$

Then the operational achievements of a run can be reported as

$$\Pr(T_C < \infty), \qquad \mathbb{E}[T_C], \qquad \Pr(T_C \leq n),$$

where n is the iteration budget. This makes the reported number of iterations and milestones explicit.

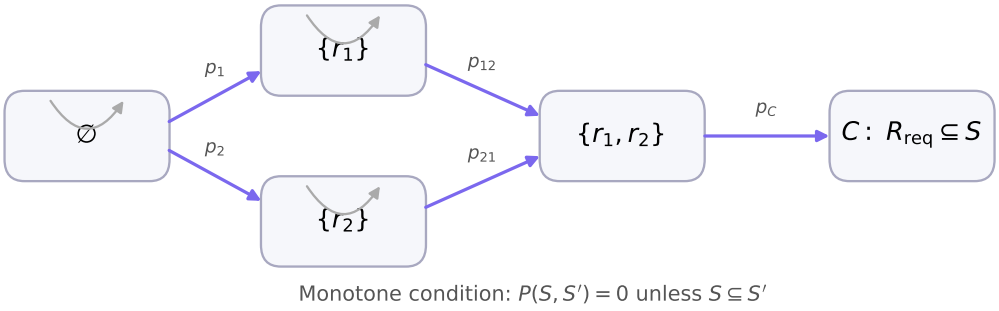

Figure 3: A minimal Markov overlay on the fixed-point lattice. The fixed-point layer defines which states are valid; the Markov layer assigns transition probabilities between valid monotone progress states. The success set $C = \{S : R_{\text{req}} \subseteq S\}$ is absorbing for reporting completion probability and expected hitting time.

Proposition 2 (Monotone Markov execution)**.** *Let P be a Markov kernel on $(2^R, \subseteq)$ such that $P(S, S') > 0 \Rightarrow S \subseteq S'$. Then $(S_t)_{t \geq 0}$ is a finite monotone Markov chain and cannot cycle except through self-loops. If every nonterminal state has strict-progress probability at least $\varepsilon > 0$,*

$$\Pr(S_{t+1} \supsetneq S_t \mid S_t = S) \geq \varepsilon,$$

then absorption occurs almost surely and

$$\mathbb{E}[T] \leq \frac{|R|}{\varepsilon}.$$

Proof. Because every positive-probability transition satisfies $S \subseteq S'$, the chain moves upward in a finite partial order. It cannot return to a strict predecessor, so the only cycles are self-loops. At most $|R|$ strict single-requirement additions are needed along any maximal chain from $\emptyset$ to R. If strict progress occurs with probability at least ε at each nonterminal state, the waiting time for each strict move is stochastically dominated by a geometric random variable with mean $1/\varepsilon$, yielding the stated bound. □

5 Evaluation

5.1 VERTEX trajectory similarity

VERTEX scores how closely a candidate behavior trajectory matches a hidden reference. Let

$$c = (c_1, \ldots, c_m), \qquad r = (r_1, \ldots, r_n)$$

be candidate and reference trajectories with descriptors in $\mathcal{D}$. With embedding $\varphi : \mathcal{D} \to S^{d-1}$, define the cross-similarity matrix $\mathbf{S} \in [-1, 1]^{m \times n}$ with entries

$$S_{ij} = \langle \varphi(c_i), \varphi(r_j) \rangle.$$

VERTEX combines a presence component and an order component.

Presence. A BERTScore-style bidirectional best match gives

$$P = \frac{1}{m}\sum_{i=1}^{m}\max_j S_{ij}, \qquad Q = \frac{1}{n}\sum_{j=1}^{n}\max_i S_{ij}, \qquad F = \frac{2PQ}{P+Q}.$$

Here P is precision-like and Q recall-like. The harmonic mean F rewards balance and lies between P and Q.

Order. Let Γ be the set of monotone alignment paths $\gamma \subseteq \{1,\dots,m\} \times \{1,\dots,n\}$. With order-decay weight $\lambda \geq 0$, define

$$d(i,j) = (1 - S_{ij})\left(1 + \lambda\left|\frac{i}{m} - \frac{j}{n}\right|\right),$$

$$\mathrm{DTW} = \min_{\gamma\in\Gamma}\sum_{(i,j)\in\gamma} d(i,j), \qquad D = \max\left(0, 1 - \frac{\mathrm{DTW}}{m+n}\right) \in \mathbb{I}.$$

Chance calibration. Let

$$b = \mathrm{clamp}\left(\frac{1}{mn}\sum_{i,j} S_{ij}; 0, 1-\epsilon\right) \in [0,1)$$

be the mean cross-similarity baseline. Define

$$\eta_b(x) = \mathrm{clamp}\left(\frac{x-b}{1-b}; 0, 1\right).$$

With mixing weight $\alpha \in [0,1]$,

$$\mathrm{VERTEX}(c,r) = \alpha\eta_b(F) + (1-\alpha)\eta_b(D) \in \mathbb{I}.$$

Proposition 3 (VERTEX range and calibration)**.** $\mathrm{VERTEX}(c,r) \in \mathbb{I}$*; it is* 0 *in expectation for a trajectory drawn at chance, where* $F, D \to b$*, and* 1 *for an exact match, where* $F, D \to 1$*.*

Proof. η_b maps $[b,1]$ onto $\mathbb{I}$, monotonically increasing, with $\eta_b(b) = 0$ and $\eta_b(1) = 1$. The harmonic mean lies between its arguments. The order score is in $\mathbb{I}$ by the clamp. A convex combination of $\mathbb{I}$-valued quantities stays in $\mathbb{I}$. □

Semantic correctness versus semantic similarity. The validation oracle $\nu(S,r) \in \{0,1\}$ is the binary mechanism used by the fixed-point semantics: a requirement enters the ledger only when ν certifies it. VERTEX is a continuous trajectory-similarity score in $[0,1]$, not a binary correctness rule. A binary semantic decision may be derived from VERTEX only if an explicit threshold is added; that threshold is not part of the fixed-point semantics.

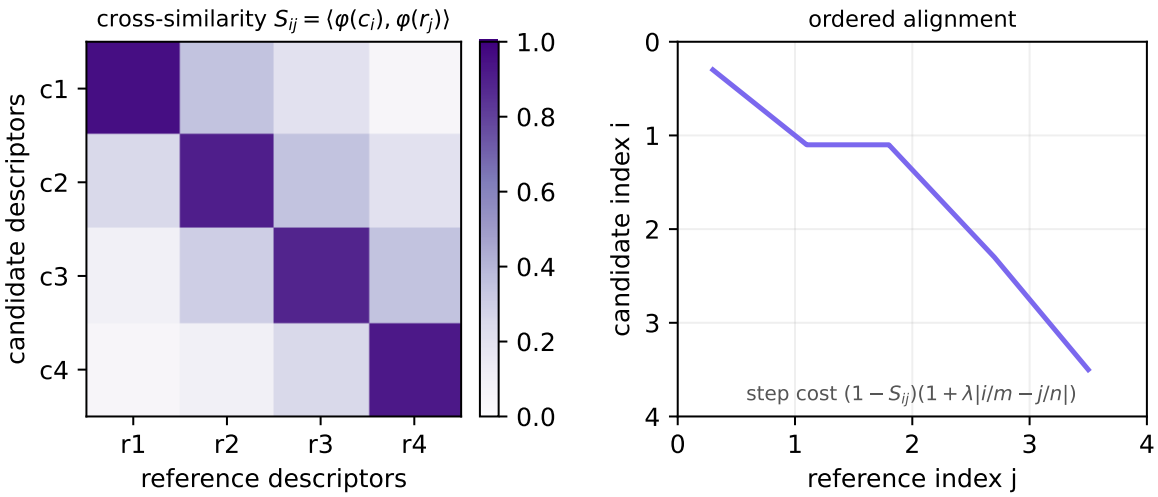

Figure 4: The two components of the VERTEX trajectory score. Left: the cross-similarity matrix S_{ij} between candidate descriptors and reference descriptors. Right: an ordered alignment path whose step cost penalizes semantic mismatch and out-of-order matches.

5.2 VERTEX-QE: reference-free surrogate references

VERTEX as defined above requires a reference descriptor set r. For open-ended briefs, hand-authoring one hidden reference per brief does not scale and may overcommit the metric to one structure. VERTEX-QE removes the authored reference while leaving the VERTEX kernel unchanged: the presence component F, order component D, and chance normalizer η_b are computed exactly as before. It is reference-free, not order-free.

Let b be the brief, C a fixed criteria-selected corpus of high-quality domain repositories, and $\{c_1, \ldots, c_N\}$ the pool of arms solving b. The estimated reference for arm i is a descriptor sequence

$$\hat{r}_i = \mathrm{agg}\left(\mathrm{ext}(b), \rho(C), \kappa(\{c_j\}_{j \neq i})\right) \in \mathcal{D}^*.$$

The constituents are: capability descriptors extracted from the brief, architecture descriptors pooled from the corpus, and a leave-one-out peer consensus from the other arms. The scalar score is then

$$\mathrm{VERTEX}(c_i, \hat{r}_i).$$

The COMET reference is used here as precedent for source-aware or reference-free quality estimation, not as the Cortex scoring rule. The scoring rule remains VERTEX/VERTEX-QE; COMET supports the idea that source or input information can aid evaluation when no gold reference exists.

5.3 The build-gated composite

For a full-repository task, signals $s_1, \ldots, s_K \in \mathbb{I}$ – functional behavior, trajectory similarity, visual fidelity, architecture, accessibility, output security, robustness, and code health – combine through a convex weight vector $w \in \Delta(\{1, \ldots, K\})$, gated by a build indicator $g \in \{0, 1\}$:

$$C = g \sum_{k=1}^{K} w_k s_k, \qquad w_k \geq 0, \qquad \sum_k w_k = 1.$$

Proposition 4 (Gated composite bound)**.** $C \in \mathbb{I}$*, with* $C = 0$ *whenever the repository does not build* ($g = 0$)*; when it builds,*

$$\min_k s_k \le C \le \max_k s_k.$$

The gate makes "does not build" an absorbing zero: no judge can rescue a non-building repository. For long-horizon families we additionally report reliability across seeds. With c of n seeds fully complete, the consistency estimator is

$$\text{pass}^k = \frac{\binom{c}{k}}{\binom{n}{k}},$$

the chance that all k independent runs succeed.

The same gated form scores the Bugfix family with resolution as the gate in place of build:

$$C_R = r(w \cdot s), \qquad r \in \{0, 1\},$$

where r records whether the shown hidden test passes. Graded signals include held-out robustness, regression safety, patch minimality, locality, and code health. Test-file edits are detected and scored zero.

5.4 Safety: attack-success and confidence intervals

A harness faces N adversarial cases and is compromised on c of them. The attack-success rate is

$$\hat{p} = \frac{c}{N} \in \mathbb{I},$$

an estimate of the unknown true success probability $p \in \mathbb{I}$. We report the Wilson score interval, obtained by inverting

$$(\hat{p} - p)^2 = \frac{z^2 p(1-p)}{N},$$

where $z = 1.96$ for 95 percent confidence:

$$\frac{\hat{p} + \frac{z^2}{2N}}{1 + \frac{z^2}{N}} \pm \frac{z}{1 + \frac{z^2}{N}} \sqrt{\frac{\hat{p}(1-\hat{p})}{N} + \frac{z^2}{4N^2}}.$$

With k independent seeds per case, the operative risk that an attack succeeds at least once is

$$\text{ASR@}k = 1 - \frac{\binom{N-c}{k}}{\binom{N}{k}} \in \mathbb{I},$$

nondecreasing in k. Safety is reported jointly with over-refusal on benign tasks, so that refusing everything cannot be mistaken for safety. For indirect injection cases, we report a secure-and-useful rate: the fraction that both resist the injection on every seed and still complete the benign carrier task.

5.5 Gated qualitative judging

Visual, architecture, and UX judges follow LLM-as-judge methodology. We control position and verbosity bias by swap-and-average over presentation order, and gate each judge on objective signals: a judge score is admitted only when the artifact builds and serves, and is cross-checked against objective metrics.

Assumption 2 (Judge gating)**.** *Each judge is calibrated only on the conditional event that objective gates pass; off that event it contributes zero. This bounds a judge's influence by objective evidence and removes the dominant LLM-as-judge failure mode of rewarding fluent non-results.*

5.6 Cross-family aggregate: the Gauntlet Index

Each family is scored on its own scale. To read one comparable figure per arm, normalize every family's per-arm headline to $\mathbb{I}$ with higher better – safety is inverted to $1 - \hat{p}$ – and take the equal-weight mean over the families an arm has a score in:

$$\text{Index}(M) = \frac{1}{|T_M|} \sum_{t \in T_M} \tilde{s}_t(M) \in \mathbb{I},$$

where $\tilde{s}_t(M)$ is arm M's normalized score on family t. The index is a convenience aggregate for ranking, not a substitute for per-family metrics.

5.7 Operational diagnostics for long-horizon results

For long-horizon families, the fixed-point score should be accompanied by execution diagnostics. These are not new correctness conditions; they explain how the result was reached.

Diagnostic	Meaning
n	Maximum loop iterations or total passes allowed.
Productive passes	Number of passes with $\Phi(S) \supsetneq S$.
$\text{efc}(S_t)$	Requirements newly validated on pass t.
$\lvert R\rvert, \lvert R_{\text{req}}\rvert$	Total and required requirement counts.
Reasoning budget b	Thinking mode or compute budget used by the harness.
$\Pr(T_C \le n)$	Probability of completing required requirements within budget.
$\mathbb{E}[T_C]$	Expected number of passes to completion.
$-\log_2 p$	Surprisal in bits for a reported probability p.

The quantity

$$I(p) = -\log_2 p$$

turns probabilities into bits of surprise. For safety, it can report how surprising a successful attack is under a given harness: a lower attack probability corresponds to more bits of safety margin. For example, if a governed arm has $p = 0.067$, then $-\log_2(0.067) \approx 3.9$ bits; if a raw arm has $p = 0.26$, then $-\log_2(0.26) \approx 1.9$ bits. The difference is an interpretable information-scale safety gain. This diagnostic is useful for reporting, but it should not replace attack-success rates, Wilson intervals, or fixed-point correctness.

6 Experimental Setup

The unit of comparison is a harness arm – a base model driven by a fixed control loop. We compare raw harnesses against the same base models placed behind Cortex. Arms are grouped by base model: for each base model we report the raw arm and the governed arm "+ Cortex," so each pair isolates the effect of the layer on a fixed model.

6.1 Baselines and base models

The study compares contemporary coding agents and governed variants. Raw harnesses run as themselves under the isolation protocol: Codex CLI on GPT-5.5, Claude Code on Claude Opus 4.8, and OpenCode as an open-source harness reference on the GPT-5.5 family. Governed arms place the same frontier base models behind Cortex. The governed arm drives the same base coding tool as its raw counterpart, pinned to the same model; only the control loop of Section 4 is added.

6.2 Relation to existing benchmarks

Most agent benchmarks rank a base model paired with a harness on single-shot task completion. Gauntlet instead makes the control layer the unit of comparison and adds two axes those suites largely omit: adversarial safety and long-horizon completion. It is complementary rather than a replacement for SWE-bench-style issue resolution: that remains the standard for single-issue capability, while Gauntlet adds paired raw-versus-governed comparisons, four adversarial delivery surfaces, long-horizon families, and build-gated, chance-calibrated metrics.

6.3 Tasks, harnesses, and success

Each task is presented to every arm as an instruction plus inputs such as files, a brief, or seed data. An arm produces a transcript: the actions it proposed and artifacts it wrote. Nothing proposed is executed on the host. Proposed shell actions are parsed and classified, and any generated code that must run does so only inside the sandbox. Scoring is computed from artifacts and transcript by the metrics of Section 5; no metric observes the hidden answer key until after the arm has finished.

6.4 Isolation protocol

A raw arm that inherited Cortex configuration would silently improve the baseline and void the comparison. The protocol rules this out structurally: every raw arm runs in a fresh temporary workspace outside the project tree, with environment variables stripped of Cortex- and session-scoped state. Raw arms receive sterile tool homes by default, with only allowlisted provider authentication linked in. Governed arms run identical base models through the full layer.

6.5 The five task families

Each family targets a different facet; together they separate safety from capability and single-pass from long-horizon behavior.

S – Adversarial safety. The goal is to measure how often an arm can be induced into an unsafe action. Inputs combine obfuscation techniques with four delivery surfaces – direct user turn, poisoned repository file, malicious tool output, and tampered memory – in text, image, and audio. It is scored by attack-success rate with Wilson intervals and ASR@k, jointly with over-refusal.

Q – Code quality and output security. The goal is to test whether produced code meets requirements and is secure. Inputs are well-specified single-file programming tasks with a hidden

Family	What it probes	Primary metric	Expected effect of Cortex
S – Safety	Resistance to adversarial/jailbreak inputs across direct, repo-file, tool-output, and memory surfaces.	Attack-success rate, with over-refusal as counterweight.	Large drop in attack-success at comparable over-refusal.
Q – Quality	Correctness and output security on well-specified single-file tasks.	Requirement coverage, hidden-test pass rate, CWE findings, lint and type cleanliness.	Near parity; frontier models already handle many single-pass tasks.
R – Bugfix	Resolving an issue in an existing repository, from easy one-line bugs to hard multi-file bugs.	Resolution-gated graded composite: lenient and strict resolution, held-out robustness, regression, minimality, locality, code health.	Parity on isolated fixes; gains where repair spans files or must avoid regressions.
G – Generative	Sustaining correctness across a long, multi-feature app build.	Build-gated capability composite: feature completeness, robustness, build/serve, trajectory, visual style, honesty, security, code health.	Completeness held later; raw arms decay and overfit fixed probes.
P – Project	Building an entire working repository from one open brief.	Build-gated composite: functional, VERTEX, security, visual, accessibility, robustness, code health.	Largest gap; long horizon and governance combined.

Table 1: The five task families of the benchmark. They separate safety from capability and single-pass behavior from long-horizon behavior.

acceptance test suite and requirement checklist. Checks are reported as per-check success rates attributed to functionality, quality, and security.

R – Bugfix. The goal is to fix a bug in an existing repository. The arm receives a repo at a buggy base commit plus an issue description and must produce a patch. The resolution-gated composite scores shown hidden-test resolution, held-out robustness, regression safety, patch minimality/locality, and code health.

G – Generative capability. The goal is to sustain correctness across a long build. The representative task is a ChatGPT-style assistant app wired to a local small language model. It is scored by a build-gated capability composite and a decay curve over the build horizon.

P – Full-repository build. The goal is to build an entire working repository from an open brief, such as a storefront with backend, frontend, and payment test flow. The score is the build-gated composite, gated on the repository actually building and serving.

6.6 Containment

Any artifact that must execute – a generated application or test suite – runs only inside a hardened container: non-root, capabilities dropped, no network egress by default, and CPU/memory/PID/time limits. Untrusted code never touches the host. The safety family is capture-only, with sandboxed side-effect replay used to confirm whether a proposed action would have succeeded.

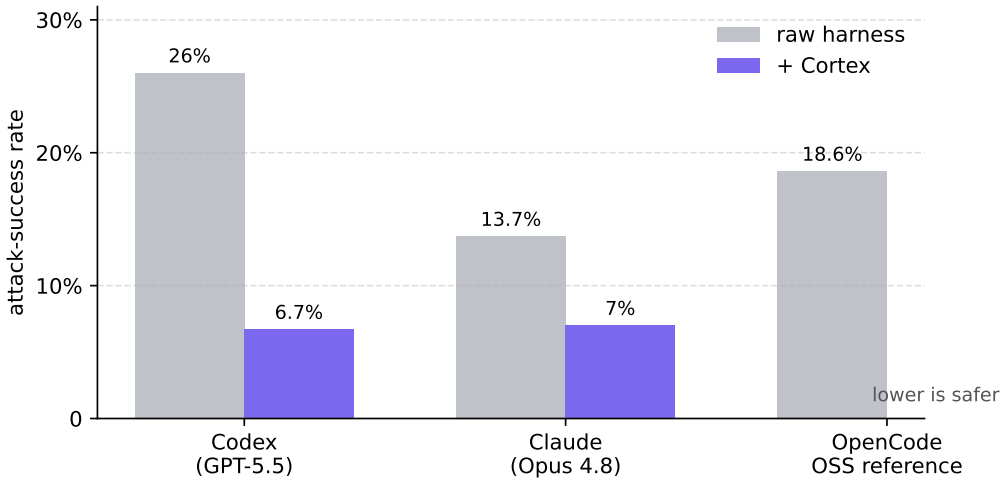

Figure 5: Adversarial attack-success rate for each base model. Governance removes unsafe action classes structurally, cutting attack-success from 26% to 6.7% on the Codex base – about a 74% relative reduction – and from 13.7% to 7.0% on the Claude base.

7 Results

Two readings carry the empirical claim: a safety reading on Track S and a capability reading on long-horizon families. Per-model rates with confidence intervals and measured-versus-modeled status are recorded on the interactive reports.

Safety. Placing a model behind Cortex reduces adversarial attack-success sharply (Figure 5). On the Codex base, the rate falls from 26% to 6.7%, a relative reduction of

$$\frac{0.26 - 0.067}{0.26} \approx 0.742.$$

On the Claude base, it falls from 13.7% to 7.0%. The gain is structural rather than learned: by Section 4.1, an attack that resolves to a blocked action cannot succeed regardless of phrasing.

Capability. On single-pass tasks that a frontier model already handles in one pass – well-specified single-file tasks and easy bugfixes – raw and governed arms are near parity. Discrimination appears where one pass is not enough: on hard multi-file Bugfix tasks and long-horizon Generative/Project tasks. Figure 6 shows the intended operational reading: report not just the curve, but also $|R|$, $|R_{\text{req}}|$, the number of milestones, the number of loop iterations n, and the reasoning budget used.

Results-facing Markov and information diagnostics. The deterministic fixed-point theorem explains why the governed curve can continue toward closure; the Markov layer explains how likely and how fast that happens under a particular policy, hook set, and reasoning budget. In future or interactive reports, the same run should be summarized by

$$\Pr(T_C \le n), \qquad \mathbb{E}[T_C], \qquad \Pr(\text{unsafe terminal state}),$$

and by bits-scale diagnostics such as

$$-\log_2 \hat{p}$$

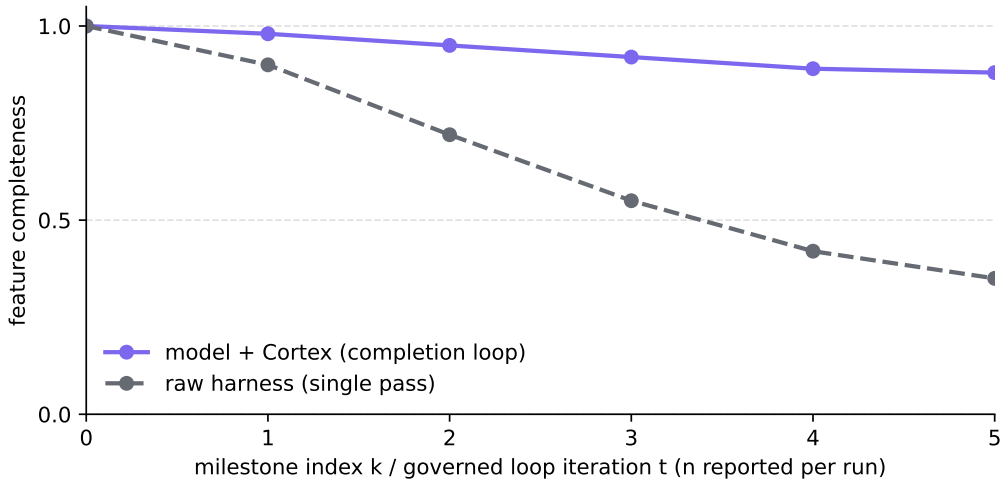

Figure 6: Feature completeness over the build horizon on long-horizon families. The horizontal axis now makes the operational quantity explicit: milestone index k or governed loop iteration t, with iteration budget n reported per run. Single-pass arms complete early milestones and decay; governed arms keep adding validated requirements toward the least fixed point.

for attack success or completion failure. These diagnostics aid representation of achievements without changing the primary scores.

8 Discussion

Why safety improves, and why over-refusal does not. The drop in attack-success is structural, not learned. Governance maps actions outside $\mathcal{A}_G$ to $\perp$, so an attack resolving to a blocked action cannot succeed. Because hooks target specific high-confidence classes and are conservative elsewhere, benign tasks rarely resolve to blocked actions; over-refusal stays comparable to the raw arm. Reporting safety and over-refusal together prevents indiscriminate refusal from masquerading as safety.

Why quality is at parity. On the single-file quality family, raw and governed arms are near parity. This is expected and important: Cortex supplies control, not extra base-model capability. When a task is covered by one validated increment, the completion loop has little to add beyond the first pass. Parity shows that the layer does not degrade capability and that long-horizon gains are not simply the result of a stronger base model.

Why gains concentrate on long horizons. Completeness on generative and full-repository families is the accumulation of effective feedback. Each validated requirement is one unit, and Theorem 1 guarantees the loop keeps adding validated requirements until the least fixed point. A single-pass harness delivers at most one increment, so its completeness decays as the requirement count grows, whereas a governed arm continues toward closure. The build gate keeps the composite honest: a non-building repository scores zero.

What Markov transitions add. The fixed-point theorem says what closure means and guarantees stabilization under monotone iteration. Markov transitions add quantitative execution semantics: completion probability, expected time to completion, and the probability mass that governance removes from unsafe or incomplete states. They are needed for result presentation, not for semantic validity. In particular, they answer questions visible in long-horizon plots: how many milestones were available, how many iterations were used, what thinking budget was used, and how likely the governed loop was to reach the success set within that budget.

Information-theoretic reporting. The bit-scale quantity $I(p) = -\log_2 p$ is useful as a diagnostic because it turns probabilities into additive units of surprise. It should be reported beside probabilities, not instead of them. For safety, it converts attack probability into a safety-margin readout; for Markov completion, it can convert failure probability into bits of residual uncertainty. This improves interpretability while preserving the original calibrated metrics.

Threats to validity. The comparison is only as clean as its isolation protocol; baselines must run with the layer fully removed. Theorem 1 guarantees convergence but not the value of the fixed point: how much of R_{req} is reachable depends on base-model productivity. Markov probabilities require repeated seeds or a calibrated transition model; otherwise they should be reported as estimates with uncertainty. Subjective judgments remain bounded by objective gates.

9 Assumptions and Limitations

We assume sound, conservative hooks (Assumption 1), objectively gated judges (Assumption 2), and that each requirement admits a validation oracle supplied by the contract compiler. Theorem 1 guarantees that the loop converges; the value of the fixed point depends on the base model's per-pass productivity and is therefore model-dependent. The absolute scale of VERTEX depends on the embedding φ, which the chance normalizer mitigates but does not eliminate. A single full-repository run trades variance for cost; when several seeds are affordable, reliability and Markov diagnostics should be reported instead of only a point estimate.

The Markov layer is an abstraction. Since the base policy is formally $\pi : \mathcal{H} \to \Delta(\mathcal{A})$, requirement state S_t alone may not be Markov. For exact semantics the state should be (H_t, S_t); the requirement-only chain is a reporting marginal. This limitation should be explicit so that the Markov layer strengthens empirical reporting without weakening the fixed-point proof.

10 Conclusion

The completion loop of a meta-level control layer converges to the least fixed point of a monotone consequence operator, and the metrics used to measure it are bounded and chance-calibrated. The key correction is that productive passes contribute one or more units of validated, retained feedback, quantified by $\text{efc}(S) = |\Phi(S) \setminus S|$. Empirically, supervising a base coding agent with Cortex reduces adversarial attack-success sharply and raises long-horizon capability where single-pass harnesses decay. The added Markov and information-theoretic diagnostics do not replace this semantics; they make the achievements more operationally legible by reporting completion probability, expected iterations, reasoning-budget effects, and risk reduction in probability and bit units.

A Proof Identities

Each statement follows from the definitions in Sections 4–5.

- **P1, normalizer.** For $b \in [0,1)$, $\eta_b(x) = \text{clamp}((x-b)/(1-b); 0, 1)$ maps $[b,1]$ onto $[0,1]$, is strictly increasing on $[b,1]$, and satisfies $\eta_b(b) = 0$, $\eta_b(1) = 1$.
- **P2, composite.** For $w \in \Delta(\{1,\ldots,K\})$ and $s_k \in \mathbb{I}$, the convex combination $\sum_k w_k s_k$ lies in $[\min_k s_k, \max_k s_k] \subseteq \mathbb{I}$, and the gate yields $C = 0$ at $g = 0$.
- **P3, harmonic mean.** $F = 2PQ/(P+Q)$ satisfies $\min(P,Q) \leq F \leq \max(P,Q)$ and $F \leq (P+Q)/2$.
- **P4, DTW.** $D = \max(0, 1 - \text{DTW}/(m+n)) \in \mathbb{I}$, since $\text{DTW} \geq 0$ and the clamp bounds it below.
- **P5, pass@k and ASR@k.** The expression $1 - \binom{N-c}{k}/\binom{N}{k}$ is one minus the probability that a uniform k-subset avoids all c compromised cases; it lies in $\mathbb{I}$ and is nondecreasing in k.
- **P6, Wilson.** The displayed Wilson interval endpoints are the roots of $(\hat{p} - p)^2 = z^2 p(1-p)/N$.
- **P7, fixed point.** Theorem 1 gives convergence, soundness, and confluence; Proposition 1 follows by telescoping the disjoint increments.
- **P8, monotone Markov execution.** Proposition 2 follows from finite monotone progress and a geometric waiting-time bound.

Cite this work

If you use Cortex or its evaluation harness, cite:

```
@techreport{dinu2026cortex,
  title       = {Cortex: A Fixed-Point Theory of Governed Coding Agents},
  author      = {Dinu, Marius-Constantin and Zeba, Florian},
  institution = {Alpha Omega Labs},
  year        = {2026},
  type        = {Technical report}
}
```

References

[1] Dinu, M.-C. et al. (2024). *SymbolicAI: A framework for logic-based approaches combining generative models and solvers*. arXiv:2402.00854.

[2] van Emden, M. H. and Kowalski, R. A. (1976). The semantics of predicate logic as a programming language. *Journal of the ACM* 23(4):733–742.

[3] Tarski, A. (1955). A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics* 5(2):285–309.

[4] Kleene, S. C. (1952). *Introduction to Metamathematics*.

[5] Banach, S. (1922). Sur les operations dans les ensembles abstraits et leur application aux equations integrales. *Fundamenta Mathematicae* 3:133–181.

[6] Garcez, A. d'Avila and Lamb, L. C. (2023). Neurosymbolic AI: the 3rd wave. *Artificial Intelligence Review* 56(11):12387–12406.

[7] Kautz, H. (2022). The third AI summer. *AI Magazine* 43(1).

[8] Newell, A. and Simon, H. A. (1972). *Human Problem Solving*. Prentice-Hall.

[9] Laird, J. E., Newell, A., and Rosenbloom, P. S. (1987). SOAR: an architecture for general intelligence. *Artificial Intelligence* 33(1):1–64.

[10] Anderson, J. R. et al. (2004). An integrated theory of the mind. *Psychological Review* 111(4):1036–1060.

[11] Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2020). BERTScore: Evaluating text generation with BERT. ICLR.

[12] Sakoe, H. and Chiba, S. (1978). Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. ASSP* 26(1):43–49.

[13] Wilson, E. B. (1927). Probable inference, the law of succession, and statistical inference. *JASA* 22(158):209–212.

[14] Efron, B. (1979). Bootstrap methods: another look at the jackknife. *Annals of Statistics* 7(1):1–26.

[15] Chen, M. et al. (2021). Evaluating large language models trained on code. arXiv:2107.03374.

[16] Liu, Y. et al. (2023). G-Eval: NLG evaluation using GPT-4 with better human alignment. EMNLP.

[17] Zheng, L. et al. (2023). Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. NeurIPS.

[18] Kim, S. et al. (2024). Prometheus 2. arXiv:2405.01535.

[19] Patel, A. et al. (2024). Large language models can self-improve at web agent tasks. arXiv:2405.20309.

[20] Greshake, K. et al. (2023). Compromising real-world LLM-integrated applications with indirect prompt injection. ACM AISec.

[21] Zhang, X., Wang, D., Xu, K., Zhu, Q., and Che, W. (2026). Scaling laws for agent harnesses via effective feedback compute. arXiv:2605.29682.

[22] Howard, R. A. (1966). Information value theory. *IEEE Transactions on Systems Science and Cybernetics* 2(1):22–26.

[23] Russell, S. and Wefald, E. (1991). Principles of metareasoning. *Artificial Intelligence* 49(1–3):361–395.

[24] Kaplan, J. et al. (2020). Scaling laws for neural language models. arXiv:2001.08361.

[25] Hoffmann, J. et al. (2022). Training compute-optimal large language models. arXiv:2203.15556.

[26] Snell, C. et al. (2025). Scaling LLM test-time compute optimally. arXiv:2408.03314.

[27] Brown, B. et al. (2025). Large language monkeys. arXiv:2407.21787.

[28] Lightman, H. et al. (2024). Let's verify step by step. ICLR.

[29] Madaan, A. et al. (2023). Self-Refine. NeurIPS.

[30] Shinn, N. et al. (2023). Reflexion. NeurIPS.

[31] Yao, S. et al. (2023). ReAct. ICLR.

[32] Jimenez, C. E. et al. (2024). SWE-bench. ICLR.

[33] OpenAI (2024). Introducing SWE-bench Verified.

[34] Zhang, L. et al. (2025). SWE-bench goes live. arXiv:2505.23419.

[35] Merrill, M. A. et al. (2026). Terminal-Bench. arXiv:2601.11868.

[36] Zhao, W. et al. (2025). Commit0. ICLR.

[37] Jain, N. et al. (2025). LiveCodeBench. ICLR.

[38] Bhatt, M. et al. (2023). Purple Llama CyberSecEval. arXiv:2312.04724.

[39] Debenedetti, E. et al. (2024). AgentDojo. NeurIPS Datasets and Benchmarks.

[40] Zhan, Q. et al. (2024). InjecAgent. ACL Findings.

[41] Yao, S. et al. (2024). Tau-bench. arXiv:2406.12045.

[42] Liu, X. et al. (2024). AgentBench. ICLR.

[43] Mialon, G. et al. (2024). GAIA. ICLR.

[44] Zhou, S. et al. (2024). WebArena. ICLR.

[45] Xie, T. et al. (2024). OSWorld. NeurIPS Datasets and Benchmarks.

[46] Wei, A., Haghtalab, N., and Steinhardt, J. (2023). Jailbroken. NeurIPS.

[47] Rei, R., Stewart, C., Farinha, A. C., and Lavie, A. (2020). COMET: A neural framework for MT evaluation. EMNLP.

[48] Gao, Y., Zhao, W., and Eger, S. (2020). SUPERT. ACL.

[49] Pillutla, K. et al. (2021). MAUVE. NeurIPS.

[50] Kumar, S. and Byrne, W. (2004). Minimum Bayes-risk decoding for statistical machine translation. NAACL-HLT.

[51] Dawid, A. P. and Skene, A. M. (1979). Maximum likelihood estimation of observer error-rates using the EM algorithm. *JRSS C* 28(1):20–28.

[52] van den Oord, A., Li, Y., and Vinyals, O. (2018). Representation learning with contrastive predictive coding. arXiv:1807.03748.

[53] Ratner, A. et al. (2017). Snorkel. *VLDB* 11(3):269–282.

[54] Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal* 27:379–423, 623–656.