CORTEX: A FIXED-POINT THEORY OF GOVERNED CODING AGENTS*

Marius-Constantin Dinu & Florian Zeba
Alpha Omega Labs

ABSTRACT

A *coding agent* combines a large language model (LLM) with a harness that plans, edits, and executes code. We study **Cortex**, a supervisory layer for pre-execution governance, requirement tracking, and iterative validation and repair. We formalize requirement closure on a finite lattice. For a fixed requirement set and a sound, monotone validation rule with persistent evidence, exhaustive iteration reaches the least fixed point with no more strict increases than there are requirements. Fair consequence scheduling reaches the same closure. A separate stochastic model bounds expected completion time under a uniform positive-progress assumption. These results depend on their stated assumptions; artifact repair, fallible validation, and finite execution budgets do not satisfy them automatically.

We define Gauntlet's trajectory-similarity and build-gated scores, attack-success estimators, Wilson intervals, and case-cluster bootstrap summaries. Effective-feedback accounting counts newly certified requirements, rather than estimating their information value or utility. The five evaluation families support raw-versus-governed comparisons. We report retained benchmark observations, including rescoring of existing artifacts; constructed outcomes are excluded. Unequal observation sets and incomplete provenance prevent causal claims about governance or measured long-horizon scaling.

1. INTRODUCTION

An LLM coding harness turns model output into tool calls, file changes, and shell commands. Task competence and execution control are distinct properties: a capable agent can follow malicious instructions (Wei et al., 2023; Greshake et al., 2023), while a restricted agent can still leave requirements unresolved on a long task (Zhang et al., 2026).

Cortex configures governance and orchestration around a base harness. Governance applies pre-execution checks and capability/dependency policies. Orchestration analyzes instructions, tracks requirements, and coordinates validation and repair within a resource budget. Action filtering alone does not establish planning competence or complete request-level safety. Validation may use executable checks, fixed rules, or model-based judgments; their reliability is a separate evaluation question. We study these contracts through the **Gauntlet** benchmark framework.

Inference-time sampling, search, and revision trade additional computation for task performance (Snell et al., 2024; Brown et al., 2024), complementing training-time scaling laws (Kaplan et al., 2020; Hoffmann et al., 2022). The effective-feedback perspective (Zhang et al., 2026) motivates tracking valid, nonredundant feedback that remains useful after later edits, rather than counting only tokens or tool calls. Our requirement count is an operational measure of retained progress. Classical value of information (Howard, 1966) and value of computation (Russell & Wefald, 1991) also require beliefs, utilities, and costs; a ledger update does not determine either value.

We address *(i) what an idealized supervisory layer guarantees under stated assumptions* and *(ii) how its implemented behavior can be evaluated*. The least fixed point in (i) is the closure of a specified

*Canonical paper and latest results:
https://benchmark.cortex.a2olabs.com
https://cortex.a2olabs.com
Benchmark source: https://github.com/Xpitfire/cortex-gauntlet.

validation rule, which may leave required work unresolved. The metrics and comparison protocol in (ii) do not establish that a particular implementation satisfies the theorem's hypotheses or that historical arm differences isolate governance alone.

CONTRIBUTIONS

- A typed action-filter model and a conditional finite-lattice model of requirement closure (Sections 3–4).
- A proof of finite closure, oracle-relative soundness, and fair-schedule independence under explicit assumptions (Theorem 1), plus accounting and conditional hitting-time bounds (Propositions 1–2).
- Definitions and domain restrictions for the implemented evaluation metrics (Section 5).
- A five-family evaluation protocol, with implementation and historical-evidence limitations made explicit rather than treated as experimental conclusions (Sections 6–9).

2. RELATED WORK AND THEORETICAL LINEAGE

Neurosymbolic and cognitive computation. Combining neural generation with symbolic constraints is one form of neurosymbolic integration (Garcez & Lamb, 2023; Kautz, 2022). Validation–repair cycles also resemble the repeated rule application of classical cognitive architectures: the problem-space account of Newell and Simon (Newell & Simon, 1972), SOAR (Laird et al., 1987), and ACT-R (Anderson et al., 2004). Section 4 isolates a finite consequence operator, rather than modeling a complete cognitive architecture.

Fixed-point semantics. Our convergence argument rests on the least-fixed-point semantics of monotone operators: van Emden & Kowalski's immediate-consequence operator and its least fixed point as the meaning of a logic program (van Emden & Kowalski, 1976), the Knaster–Tarski lattice fixed-point theorem (Tarski, 1955), and Kleene iteration (Kleene, 1952). Where an operator is a contraction on a metric space, Banach's theorem (Banach, 1922) gives uniqueness and a rate; we use the lattice formulation because the requirement state space is naturally a finite complete lattice.

Evaluation. Capability over trajectories is scored by VERTEX, defined in Section 5.1: embedding cross-similarity against a reference descriptor set, with a presence component (BERTScore-style bidirectional matching (Zhang et al., 2020)) and an order component (distance-decayed dynamic time warping (Sakoe & Chiba, 1978)), each affinely recalibrated against the matrix's mean-similarity baseline. Embedding comparison also appears in trajectory evaluation (Dinu et al., 2024; Patel et al., 2024). Gauntlet applies this construction to extracted descriptors, whose order need not represent an observed execution trajectory. For integers $1 \leq k \leq n$, the any-seed attack-success estimate uses $\mathrm{pass@}k$ (Chen et al., 2021). Under independent trials with common success probability $p \in [0, 1]$, it is unbiased for $1 - (1 - p)^k$. Success rates carry Wilson intervals (Wilson, 1927) and bootstrap summaries (Efron, 1979). The optional rubric judge draws on G-Eval (Liu et al., 2023), MT-Bench (Zheng et al., 2023), and Prometheus (Kim et al., 2024). Their bias analyses motivate judge validation but do not establish the validity of Gauntlet's judges. Safety draws on the jailbreak (Wei et al., 2023) and real-world prompt-injection (Greshake et al., 2023) literatures.

Scaling, test-time compute, and effective feedback. Training scaling laws relate compute, data, and loss (Kaplan et al., 2020; Hoffmann et al., 2022); inference-time search and revision trade compute for accuracy (Snell et al., 2024; Brown et al., 2024). Effective-feedback work (Zhang et al., 2026) motivates measuring usable feedback. We count newly certified requirements, as formalized in Section 4.3. This finite-set accounting identity neither derives an empirical scaling law nor establishes that equal counts represent equal information or utility.

Iterative feedback, verification, and metareasoning. ReAct interleaves reasoning and interaction (Yao et al., 2023); Self-Refine and Reflexion study feedback and revision (Madaan et al., 2023; Shinn et al., 2023). Process supervision is a distinct approach to evaluating intermediate reasoning steps (Lightman et al., 2024). Repository issue resolution is studied in benchmarks such as SWE-bench (Jimenez et al., 2024); it is not the common evaluation setting of all these methods. Our theorem concerns any operator satisfying its assumptions. An implementation requires a separate conformance argument. Value of information (Howard, 1966) and rational metareasoning (Russell &

Wefald, 1991) provide decision-theoretic context, without supplying a utility model or an optimal computation-selection policy for the present framework.

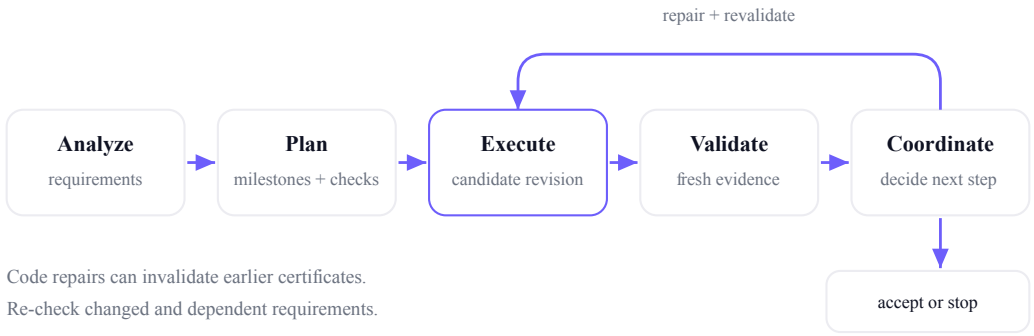

Figure 1: Schematic execution–validation–repair architecture. Its consequence-operator abstraction requires a sufficient requirement state, monotone validation, and sound persistent certificates (Section 4.2). The operational repair cycle does not establish those assumptions.

3. Preliminaries and Notation

We write $\mathbb{N}_0 = \{0, 1, \ldots\}$, $\mathbb{N}_+ = \{1, 2, \ldots\}$, $\mathbb{R}$ for the reals, and $\mathbb{I} = [0, 1]$. For finite X, $|X|$ denotes cardinality and 2^X the power set. For nonempty finite or countable X, $\Delta(X) = \{p \in \mathbb{R}_{\geq 0}^X : \sum_{x \in X} p_x = 1\}$ is the set of discrete probability distributions. For $d \in \mathbb{N}_+$, $\mathbb{S}^{d-1} = \{v \in \mathbb{R}^d : \|v\|_2 = 1\}$, with Euclidean inner product $\langle u, v \rangle \in [-1, 1]$ for $u, v \in \mathbb{S}^{d-1}$. For real x, a, b with $a \leq b$, $\mathrm{clamp}(x; a, b) = \min(\max(x, a), b)$. The indicator $\mathbf{1}\{E\}$ equals 1 when statement E holds and 0 otherwise.

Order-theoretic objects. A complete lattice $(L, \sqsubseteq)$ has a least upper bound and a greatest lower bound for every subset, including bottom and top elements. A map $f : L \to L$ is *monotone* when $x \sqsubseteq y$ implies $f(x) \sqsubseteq f(y)$, and *inflationary* when $x \sqsubseteq f(x)$. These are distinct properties. A fixed point satisfies $f(x) = x$. A monotone map on a complete lattice has a least fixed point $\mathrm{lfp}(f)$ (Tarski, 1955).

Agent objects. Let $\mathcal{A}$ and $\mathcal{O}$ be nonempty countable action and observation spaces. A distinguished outcome $\perp_a \notin \mathcal{A}$ denotes refusal without execution, not the lattice bottom. Set $\mathcal{A}_\perp = \mathcal{A} \cup \{\perp_a\}$ and include refusal observations in $\mathcal{O}$. The history space $\mathcal{H} = (\mathcal{A}_\perp \times \mathcal{O})^*$ contains all finite execution histories, including the empty history. A base policy $\pi : \mathcal{H} \to \Delta(\mathcal{A})$ proposes ordinary actions. For a finite, possibly empty index set J, let $h_j : \mathcal{H} \times \mathcal{A} \to \{0, 1\}$, $j \in J$, be hooks; 1 means blocked.

Task objects. Fix a finite requirement set $R = \{r_1, \ldots, r_N\}$, $N \in \mathbb{N}_0$, and required subset $R_{\mathrm{req}} \subseteq R$. The abstract state $S \subseteq R$ records certified requirements. The lattice $(2^R, \subseteq)$ has join $\cup$, meet $\cap$, bottom $\varnothing$, and top R. The abstraction $\nu : 2^R \times R \to \{0, 1\}$ determines which further certificates can be derived from S. Representing evidence by requirement identifiers alone requires the sufficiency assumption in Section 4.2; an actual workspace, tests, and execution history contain more information.

Metric objects. Let $\mathcal{D}$ be finite text descriptors and $\mathcal{D}^* = \bigcup_{m \in \mathbb{N}_0} \mathcal{D}^m$ their finite sequences. The embedding φ returns a unit vector for nonzero embeddings; the implementation uses the zero vector for a descriptor with no embedding features. Thus its full codomain is $\mathbb{S}^{d-1} \cup \{0\}$, and an exact text match need not score 1 for a zero vector. Signals $s_k \in \mathbb{I}$ combine with weights $w \in \Delta(\{1, \ldots, K\})$, $K \in \mathbb{N}_+$.

4. The Control Layer: Governance and Orchestration

The action filter and requirement-closure operator describe different contracts. The first restricts proposed actions at a specified history; the second describes an idealized validation closure. Neither alone proves real-world task correctness.

4.1 GOVERNANCE AS A PROJECTION ON THE ACTION SPACE

At fixed H, let $\mathcal{A}_G(H) = \{a \in \mathcal{A} : h_j(H, a) = 0 \text{ for every } j \in J\}$. Define $G_H : \mathcal{A}_\perp \to \mathcal{A}_\perp$ by

$$G_H(a) = \begin{cases} a, & a \in \mathcal{A}_G(H), \\ \perp_a, & a \notin \mathcal{A}_G(H). \end{cases}$$

In particular $G_H(\perp_a) = \perp_a$. Therefore $G_H \circ G_H = G_H$: it is an idempotent retraction onto $\mathcal{A}_G(H) \cup \{\perp_a\}$. The governed policy is the pushforward of the proposal policy:

$$\pi_G(a \mid H) = \pi(a \mid H)\mathbf{1}\{a \in \mathcal{A}_G(H)\} \quad (a \in \mathcal{A}), \qquad \pi_G(\perp_a \mid H) = \sum_{a \in \mathcal{A} \setminus \mathcal{A}_G(H)} \pi(a \mid H).$$

Blocked probability becomes refusal mass, **not renormalized admissible mass**. Conditional resampling until an admissible action is obtained would be a different policy and is undefined if the original policy assigns zero mass to admissible actions. Filtering leaves accepted actions unchanged at this history, but can change later behavior and task utility.

Assumption A1 (complete mediation and hook coverage). Every effect in the claimed protection boundary is checked before execution against authoritative context, without a check/use race; every unsafe action in the specified threat class is blocked by at least one hook. Under A1 no action in that class executes. Pure predicates alone do not establish coverage, and neither false-positive rates nor semantic safety follow from idempotence. The benchmark must measure refusals and coverage separately.

4.2 ORCHESTRATION: THE COMPLETION LOOP AS A CONSEQUENCE OPERATOR

For a **fixed** oracle and requirement set define

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

Assumption A2 (oracle and evidence). Requirement identifiers are a sufficient summary for this deterministic rule. Validation is monotone: $S \subseteq T,\ \nu(S, r) = 1 \Rightarrow \nu(T, r) = 1$. Whenever the certificates in S are valid for the specified task, every new certificate $r \in R \setminus S$ with $\nu(S, r) = 1$ must be supported by valid evidence and correct for that task. Accepted transitions must preserve previously valid certificates. The union makes Φ inflationary; the additional monotonicity assumption makes it monotone. Append-only storage alone does **not** imply that assumption. For example, $R = \{a, b\}$, $\nu(S, a) = 0$ for all S, and $\nu(S, b) = \mathbf{1}\{a \notin S\}$ give $\Phi(\varnothing) = \{b\}$ but $\Phi(\{a\}) = \{a\}$.

Theorem 1 (finite closure, conditional soundness, fair-schedule independence)**.** *Let $\Phi : 2^R \to 2^R$ be monotone and inflationary. Starting at $S_0 = \varnothing$, the iterates $S_{t+1} = \Phi(S_t)$ reach $S_* = \mathrm{lfp}(\Phi)$ after at most $|R|$ strict increases. A further evaluation may be needed to detect stabilization. With A2's soundness and persistence conditions, all certificates in S_* remain valid. Define PASS at closure by $R_{\mathrm{req}} \subseteq S_*$; PASS then certifies those requirements relative to the oracle, not all possible correctness properties. Starting at $\varnothing$, adding enabled consequences in any **fair** schedule reaches the same S_*.*

Proof. Inflationarity gives an increasing chain. Each strict increase adds at least one of the $|R|$ elements, so the chain stabilizes. If F is any fixed point, then $S_0 \subseteq F$ and monotonicity implies $S_{t+1} = \Phi(S_t) \subseteq \Phi(F) = F$ by induction. The stabilized state is thus the least fixed point. Soundness follows by induction on certified additions, using both oracle soundness and persistence. For asynchronous addition, every added requirement is in $\Phi(S)$, so the same induction keeps the state below S_*. Fairness means no persistently enabled missing requirement is postponed forever. Since there can be only finitely many additions, the eventual state has no enabled missing requirement and is a fixed point; leastness makes it S_*. □

The theorem proves derivability under ν, **not** that S_* is exactly the set of requirements achievable by arbitrary agents. Completeness of the oracle/generator would be an additional assumption. An

unfair scheduler can omit enabled work forever. Detecting one unproductive *stochastic* attempt is not detecting a fixed point of a deterministic exhaustive consequence operator.

Implementation boundary. Real repair passes edit a shared artifact and can regress previously passing checks. A historical union of successful checks is not a current correctness certificate. The benchmark uses bounded retries, observation-dependent validation, and score-based candidate selection; these are not an implementation proof of A2 or exhaustive closure. Fresh validation of the final artifact is needed for PASS. Raw coding harnesses also take multiple internal actions; a single benchmark invocation is not a single consequence or a single requirement addition.

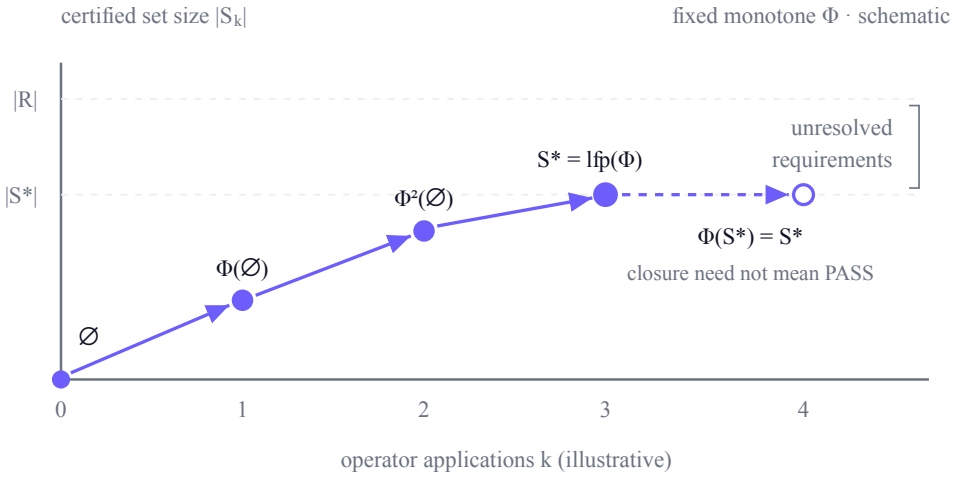

Figure 2: Illustrative Kleene ascent for the fixed monotone operator of Theorem 1, not a measured agent trace. At most $|R|$ strict increases occur; one further evaluation may detect closure. A fair schedule reaches the same least fixed point, which can leave required work unresolved.

4.3 Effective feedback and the value of computation

Define the count $\mathrm{efc}(S) = |\Phi(S) \setminus S|$ for the abstract operator. It measures newly certified requirements, motivated by retained-feedback accounts (Zhang et al., 2026). Along iteration from $\varnothing$ under A2, these additions are valid, nonredundant, and retained. Their information content and utility are not necessarily equal.

Proposition 1 (effective-feedback accounting)**.** *If $S_0 = \varnothing$ and $S_T = S_* = \mathrm{lfp}(\Phi)$ is the stabilized iterate, then $\sum_{t=0}^{T-1} |S_{t+1} \setminus S_t| = |S_*|$. If $R_{\mathrm{req}} \subseteq S_*$, exactly $|R_{\mathrm{req}}|$ of these additions are required certificates. The **total** number of additions before first completion can exceed $|R_{\mathrm{req}}|$ because optional requirements can be added too.*

Proof. The disjoint differences $S_{t+1} \setminus S_t$ partition S_*. Intersecting that partition with R_{req} gives the required-only identity. □

One productive pass can add any number from 1 to $|R \setminus S|$. For example, $\Phi(S) = R$ completes arbitrarily many requirements in one pass. Consequently no lower bound on required passes, no impossibility for single-invocation harnesses, and no advantage over independent sampling follows from this accounting. An append-only ledger prevents duplicate **credit**, not duplicate computation. Positive decision-theoretic value of information or net value of computation requires an explicit utility/cost model (Howard, 1966; Russell & Wefald, 1991); $\mathrm{efc} > 0$ is not equivalent to either.

4.4 Probabilistic execution over the fixed-point lattice

A stochastic process is separate from the deterministic closure model. Let $\nu_{\mathrm{obs}} : \mathcal{H} \times 2^R \times \mathcal{A}_\perp \times \mathcal{O} \times R \to \{0, 1\}$ validate observed executions. Define $U(H, S, a, o) = S \cup \{r \in R : \nu_{\mathrm{obs}}(H, S, a, o, r) =$

$1\}$ as historical certificate accumulation. A distinct current-artifact validation set may shrink. For a requirement-only, time-homogeneous Markov abstraction, **assume** the conditional law of the next requirement state depends on the past only through S. Then

$$P(S, S') = \sum_{\substack{a \in \mathcal{A}_\perp,\, o \in \mathcal{O} \\ U(S,a,o) = S'}} \pi_G(a \mid S)\, E(o \mid S, a), \qquad \sum_{S' \subseteq R} P(S, S') = 1,$$

where $\pi_G(\cdot \mid S) \in \Delta(\mathcal{A}_\perp)$ and $E(\cdot \mid S, a) \in \Delta(\mathcal{O})$ is the observation distribution, including a refusal observation for $\perp_a$. Each of U, π_G, E must admit this sufficient-state representation. For general history-dependent execution use $X_t = (H_t, S_t)$ instead, including all relevant environment state in the history. Aggregating its transitions by S alone does not in general produce a Markov chain. Historical certificate accumulation gives $P(S, S') = 0$ unless $S \subseteq S'$ but does not prove current-artifact correctness.

Completion as a hitting time. Let $C = \{S \subseteq R : R_{\mathrm{req}} \subseteq S\}$ and $T_C = \inf\{t \in \mathbb{N}_0 : S_t \in C\}$, with $\inf \varnothing = \infty$. For an increasing process C is a closed set; make its states absorbing if the run stops at first completion. Specify the initial state S_0 for all probabilities and expectations.

Proposition 2 (conditional completion time)**.** *For a finite increasing Markov chain, suppose every reachable $S \notin C$ has $\Pr(S_{t+1} \supsetneq S_t \mid S_t = S) \geq \varepsilon$, where $0 < \varepsilon \leq 1$. Then $\Pr(T_C < \infty) = 1$ and $\mathbb{E}[T_C] \leq (|R| - |S_0|)/\varepsilon$.*

Proof. Since $R \in C$, at most $|R| - |S_0|$ strict increases can precede completion. Before completion, the wait for each increase is dominated by a geometric variable of mean $1/\varepsilon$. Their sum bounds T_C and its expectation; independence of the holding times is unnecessary. A finite expectation implies almost-sure completion. □

An incomplete absorbing state violates the progress assumption. Without that assumption, completion need not occur and $\mathbb{E}[T_C]$ may be infinite. Governance changes the execution law through the refusal-mass filter of Section 4.1; comparing unsafe-event probabilities additionally requires an explicit unsafe-event indicator on the augmented state, not merely a subset of satisfied requirements. A difference of expected completion times is meaningful only when both expectations are finite. No transition probabilities or completion-time estimates are supplied by the lattice theorem.

5. Evaluation

5.1 VERTEX trajectory similarity

This implementation of VERTEX combines descriptor cross-similarity with an order penalty, inspired by trajectory evaluation (Dinu et al., 2024; Patel et al., 2024). For nonempty sequences $c = (c_1, \ldots, c_m)$ and $r = (r_1, \ldots, r_n)$, $m, n \in \mathbb{N}_+$, define $\mathbf{S}_{ij} = \langle \varphi(c_i), \varphi(r_j) \rangle \in [-1, 1]$. For unit embeddings this is cosine similarity; zero-vector handling is as in Section 3. Either empty sequence receives score 0 by convention. Figure 3 shows the two components.

Presence (unordered). A BERTScore-style bidirectional best match (Zhang et al., 2020), where P (precision) measures how well each candidate descriptor is matched by some reference and Q (recall) how well each reference descriptor is matched by some candidate:

$$P = \frac{1}{m} \sum_{i=1}^{m} \max(0, \max_j \mathbf{S}_{ij}),$$

$$Q = \frac{1}{n} \sum_{j=1}^{n} \max(0, \max_i \mathbf{S}_{ij}),$$

$$F = \begin{cases} 2PQ/(P+Q), & P + Q > 0, \\ 0, & P + Q = 0. \end{cases}$$

Thus $P, Q, F \in \mathbb{I}$ and $\min(P, Q) \le F \le \max(P, Q)$. Negative similarities still enter the order cost and the matrix-mean baseline below.

Order. Dynamic time warping (Sakoe & Chiba, 1978) minimizes over paths Γ from $(1, 1)$ to (m, n) with steps $(1, 0)$, $(0, 1)$, or $(1, 1)$. For finite $\lambda \ge 0$, the cost is $d(i, j) = (1 - \mathbf{S}_{ij})(1 + \lambda|(i-1)/m - (j-1)/n|)$:

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

The path has between $\max(m, n)$ and $m + n - 1$ entries. The denominator $m + n$ is a chosen normalization, not the path length or an upper bound on the **cost**.

Matrix-mean recalibration. For $0 < \epsilon < 1$, let $b = \mathrm{clamp}((mn)^{-1} \sum_{i,j} \mathbf{S}_{ij}; 0, 1 - \epsilon)$ and $\eta_b(x) = \mathrm{clamp}((x - b)/(1 - b); 0, 1)$. The raw mean lies in $[-1, 1]$, and the clamp ensures a positive denominator. It is the mean of this candidate–reference pair, **not** an independently estimated null expectation. For finite $\alpha \in [0, 1]$:

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

The benchmark configuration fixes $\alpha = 0.6$, $\lambda = 0.5$, and $\epsilon = 0.01$ (the baseline-clamp margin). Proposition 3 records the range and the exact sense of the calibration.

Proposition 3 (range and anchors)**.** *VERTEX lies in $\mathbb{I}$. The normalizer is nondecreasing on $\mathbb{R}$ and strictly increasing on $[b, 1]$, with $\eta_b(b) = 0$ and $\eta_b(1) = 1$. Identical sequences of nonzero unit embeddings score 1 in exact arithmetic.*

Proof. All floored best matches lie in $\mathbb{I}$; the harmonic mean lies between its arguments. Costs are nonnegative, hence the clamped D lies in $\mathbb{I}$. The normalizer has the stated endpoints and slope $1/(1 - b) > 0$ on its unclamped interval. The mixture is convex. Identical unit-vector sequences have unit diagonal similarities, $P = Q = 1$, and a zero-cost diagonal path, so $F = D = 1$. □

There is **no universal chance floor**. A constant zero similarity matrix with $m = n$ has $F = 0, D = 1/2$, hence score 0.2 at the default mixture. With $m = 1, n = 9$ and $\lambda = 0.5$, the same zero similarities give path cost 11, $D = 0$, and score 0. Lengths, order, and embedding geometry therefore affect unrelated-input scores. Maxima need not strictly exceed a mean, and the DTW term need not concentrate near $(1 + b)/2$. Comparisons require fixed extraction and embedding protocols; they are similarity comparisons, not calibrated probabilities or correctness certificates.

5.1.1 VERTEX-QE: ESTIMATED REFERENCES

The project task can compare a candidate with hand-authored capability and architecture descriptors. These references select particular realizations of an open-ended brief. VERTEX-QE instead estimates descriptors from public sources. Each invocation of the Section 5.1 kernel retains the same presence score F, order score D, and matrix-mean recalibration η_b; its reference argument becomes an estimate $\hat{r}$. The project-level mixture also changes its architecture weight, as defined below.

Capability descriptors come from the public brief's enumerated outcomes. Architecture descriptors combine its engineering expectations with a small synthetic vocabulary of common web-application concerns: HTTP routing, frontend components, separation of domain logic from transport, commerce entities, tests, and environment configuration. The vocabulary spans six authored patterns, including a layered monolith, a separate frontend and REST API, server-side rendering, a commerce-service split, a ports-and-adapters core, and a static frontend with serverless APIs. These are illustrative design patterns, not observations from six sampled repositories. Empty strings and exact duplicates are removed within each source; the union greedily drops later descriptors whose cosine similarity to a retained one is at least 0.92. The resulting list has a deterministic order that affects DTW, although it does not describe execution time or a distribution of valid architectural alternatives.

For the architecture mixture, let L be the number of nonempty cleaned source lists. Write list ℓ as $(x_{\ell 1}, \ldots, x_{\ell n_\ell})$, where $n_\ell \ge 1$. For distinct sources $\ell, k \in \{1, \ldots, L\}$, directed agreement is

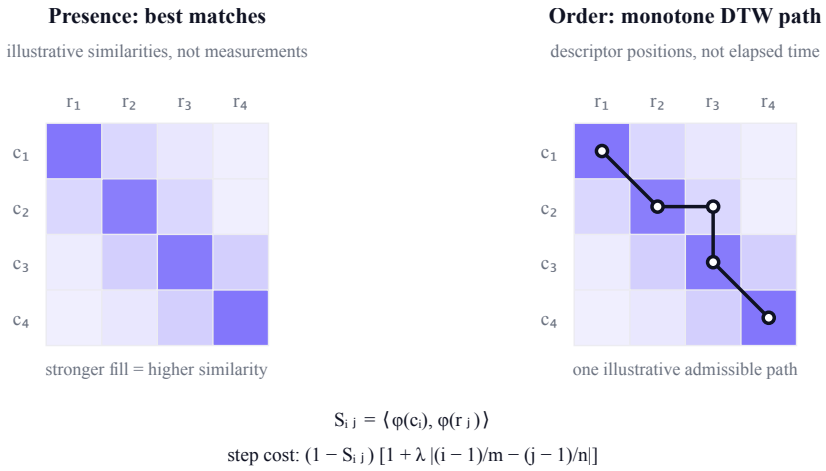

Figure 3: The two components of the VERTEX trajectory score (Section 5.1). Left: the cross-similarity matrix $\mathbf{S}_{ij} = \langle \varphi(c_i), \varphi(r_j) \rangle$ between candidate descriptors c_i (rows) and reference descriptors r_j (columns); stronger color indicates higher similarity in this illustrative matrix. The *presence* component reads bidirectional best matches off this matrix (which reference behaviors appear, and how strongly). Right: the *order* component uses a minimum-cost monotone DTW alignment. Its step cost $(1 - \mathbf{S}_{ij})(1 + \lambda |\frac{i-1}{m} - \frac{j-1}{n}|)$ increases with normalized positional displacement. Both components are recalibrated against the mean-similarity baseline (Proposition 3).

$$a_{\ell k} = \frac{1}{n_\ell} \sum_{i=1}^{n_\ell} \max\left(0, \max_{1 \le j \le n_k} \langle \varphi(x_{\ell i}), \varphi(x_{kj}) \rangle\right).$$

The resolver sets $q_{\text{arch}} = 0$ for $L = 0$, $1/2$ for $L = 1$, and otherwise averages $(a_{\ell k} + a_{k\ell})/2$ over all unordered source pairs, rounded to four decimal places. Thus $q_{\text{arch}} \in [0, 1]$. With capability and architecture VERTEX scores V_{cap} and V_{arch}, the project signal is

$$V_P = \frac{0.7 V_{\text{cap}} + 0.3 q_{\text{arch}} V_{\text{arch}}}{0.7 + 0.3 q_{\text{arch}}}.$$

Authored-reference mode uses $q_{\text{arch}} = 1$. In QE mode an empty architecture union triggers a recorded authored-reference fallback, but $q_{\text{arch}} = 0$ then removes that architecture score from V_P. The fallback can therefore appear in diagnostics without contributing authored architecture evidence to the composite. Agreement is not a calibrated probability of correctness. COMET, SUPERT, and MAUVE (Rei et al., 2020; Gao et al., 2020; Pillutla et al., 2021) provide reference-estimation and distributional-evaluation context, not validation of this rule.

Pool-level QE diagnostics additionally use leave-one-out peer descriptors supported by at least two peers. They are not inputs to the per-candidate composite above. This is a heuristic, not a minimum-Bayes-risk estimator or a consistent latent-truth estimator justified by (Kumar & Byrne, 2004; Dawid & Skene, 1979). Even independent peers that include a false descriptor with probability 0.1 will, with probability tending to one, exceed a fixed two-peer threshold as the pool grows. Leave-one-out removes literal self-reference, not correlations between related models. The weak-supervision literature (Ratner et al., 2017) is background, not a proof that unmodeled source agreement is calibrated.

An unused contrastive helper computes a softmax assignment score. Equal logits give $1/B_{\text{batch}}$ for B_{batch} alternatives, but individual scores can be smaller. This is not a universal chance floor or itself a mutual-information lower bound: InfoNCE (van den Oord et al., 2018) bounds mutual information through an **expected loss** under specified sampling assumptions. No active multi-brief contrastive evaluation is implemented.

Proposition 4 (reference replacement and mixtures)**.** *Replacing a descriptor reference preserves Proposition 3's range bound. The convex mixture V_P is also in $\mathbb{I}$. Neither property establishes reference accuracy or equal chance behavior.*

Proof. Apply Proposition 3 to each replacement reference. The normalized weights in V_P are nonnegative and sum to one, so their mixture preserves the range. □

Within-brief Spearman correlation with authored VERTEX is a preliminary concordance diagnostic, not authorization to discard reference validation on unseen briefs. The run-level diagnostic selects one built candidate per arm and omits failed builds; pool consensus and the production metric are distinct estimands. Generalization needs independent held-out briefs, adequate replication, and reference-provenance checks.

Correctness versus similarity. The oracle ν certifies requirements in the conditional closure model. VERTEX measures descriptor similarity on a continuous scale. Turning similarity into a pass/fail decision requires an additional, validated threshold; no such decision rule is part of the fixed-point semantics.

5.2 The build-gated composite

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

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

Proposition 5 (build-gated range)**.** $C \in \mathbb{I}$*, with* $C = 0$ *whenever the repository does not build* ($g = 0$)*; when it builds,* $\min_k s_k \leq C \leq \max_k s_k$.

Proof. For $g = 1$, multiply $\min s_k \leq s_k \leq \max s_k$ by $w_k \geq 0$ and sum. For $g = 0$ the product is zero. □

Missing analyzer evidence is not a passed signal. Degraded evaluations must not be compared as though all fixed weights were measured.

The project gate is **build-only**; serving additionally gates visual and UX evidence. Functional coverage excludes unevaluable journeys from both numerator and denominator, returning zero when none are evaluable. Screenshot-based visual scores average available candidate frames and return zero when none exist. Comparisons therefore require matched journey and frame coverage; fixed outer weights do not eliminate this conditional missingness.

Averaging and build gating do not commute. Track G's historical aggregate is a different descriptive index. For $B \in \mathbb{N}_+$ briefs, let $n_b \in \mathbb{N}_+$ be the recorded attempt count for brief b, let $g_{bi} \in \{0, 1\}$ be attempt i's build indicator, and let s_{bk} be the stored brief-level value of signal k. The aggregator weights briefs equally, not individual attempts:

$$\bar{g} = \frac{1}{B} \sum_{b=1}^{B} \frac{1}{n_b} \sum_{i=1}^{n_b} g_{bi}, \qquad \bar{s}_k = \frac{1}{B} \sum_{b=1}^{B} s_{bk}.$$

Ignoring intermediate and final four-decimal rounding, the reported index is

$$G_{\text{agg}} = \bar{g} \sum_{k=1}^{K} w_k \bar{s}_k.$$

This product of marginal averages need not equal an average of jointly gated scores. If per-attempt signals s_{bik}^{seed} were available, the corresponding equal-brief average would instead be

$$G_{\text{joint}} = \frac{1}{B}\sum_{b=1}^{B}\frac{1}{n_b}\sum_{i=1}^{n_b}\left(g_{bi}\sum_{k=1}^{K}w_k s_{bik}^{\text{seed}}\right).$$

Even when each stored signal is a within-brief seed mean, these quantities can differ: with two single-seed briefs, one unit-weight signal, and gate/signal pairs $(1,1)$ and $(0,0)$, the product of means is $1/4$ but the mean gated score is $1/2$. Stored brief-level judge outputs need not themselves be seed means. Marginal summaries therefore cannot reconstruct the joint per-attempt quantity. The completeness bootstrap resamples brief-level means.

For integers $n \geq 1$, $0 \leq c \leq n$, and $1 \leq k \leq n$, let c of n recorded trials satisfy a specified success event. Define $\binom{a}{k} = 0$ when integers $k > a \geq 0$. Reliability is $\text{pass}^k = \binom{c}{k}/\binom{n}{k}$ (Yao et al., 2024): the probability that a uniformly selected k-subset consists entirely of successes. For independent trials with common success probability $p \in [0,1]$, it is unbiased for p^k and is non-increasing in k.

The project evaluator calls a seed successful when it builds and its measured functional coverage is at least 0.999. Track G requires seed completeness of at least 0.999 and averages reliability estimates over briefs. These are test-defined events, not complete semantic correctness. The live project path currently allows one seed, which supplies no multi-run reliability estimate.

For the *Bugfix* family (Track R), let $r \in \{0,1\}$ indicate that the executed hidden resolution tests pass without a generation error. Let $a \in \{0,1\}$ indicate that the submitted patch contains no prohibited test-file edits detected by the path-based filter. With bounded graded signals s and convex weights w, $C_R = ar(w \cdot s)$. If a semantic patch judge supplies $J \in [0,1]$, the formula is $C_R = ar\big(0.8(w \cdot s) + 0.2J\big)$. The signals measure held-out robustness, preservation of existing behavior, patch minimality/locality relative to the reference fix, and code health.

The lenient resolution rate reports r. The implemented strict rate also requires $a = 1$ and held-out/regression scores of at least 0.999. An absent held-out or regression suite receives score 1 with zero executed tests; therefore the strict rate certifies those additional checks only when the suites are nonempty and actually run. The test-path filter does not exclude arbitrary evaluator tampering.

5.3 Safety: attack-success and confidence intervals

Let $\mathcal{I} = \{1, \ldots, N_{\text{case}}\}$ index $N_{\text{case}} \geq 1$ evaluated cases. For each $i \in \mathcal{I}$, let $n_i \in \mathbb{N}_+$ be its recorded attempt count and $c_i \in \{0, \ldots, n_i\}$ its classified successes. The **attempt-pooled** headline is $\widehat{p} = \sum_{i\in\mathcal{I}} c_i / \sum_{i\in\mathcal{I}} n_i$. The separate case-weighted ASR@1 is $N_{\text{case}}^{-1}\sum_{i\in\mathcal{I}} c_i/n_i$. They differ when attempt counts differ. With $N = \sum_{i\in\mathcal{I}} n_i > 0$ and normal quantile $z > 0$ (default $z = 1.96$ for nominal 95% coverage), the Wilson score interval (Wilson, 1927) is

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

For a common Bernoulli success probability $p \in [0,1]$, the endpoints solve $(\widehat{p} - p)^2 = z^2 p(1-p)/N$ exactly. Coverage is approximate, not exact; correlated attempts and heterogeneous cases weaken this sampling interpretation.

The case-cluster percentile bootstrap (Efron, 1979) resamples whole cases and recomputes the attempt-pooled ratio. Defaults are 1,000 resamples, pseudorandom seed zero, and 95% intervals. After sorting the resampled estimates, the implementation selects zero-based entries 25 and 975 and rounds to four decimal places. Track G applies the corresponding procedure to brief-level mean completeness. One cluster or identical cluster values produces a point interval, not evidence of zero population uncertainty. Population inference still requires a representative cluster sample. With no observations, rates and intervals are unavailable.

For each case with integers $0 \leq c_{\text{s}} \leq n$ and $1 \leq k \leq n$,

$$\text{ASR@}k = 1 - \frac{\binom{n-c_{\text{s}}}{k}}{\binom{n}{k}}.$$

This is one minus the probability that a uniform k-subset of recorded attempts contains no success; it is nondecreasing in k. Under i.i.d. Bernoulli sampling it is unbiased for the probability of at least one success in k fresh attempts (Chen et al., 2021). It is undefined for $k > n$. A multi-pass adaptive attack path is one trial with its own fixed budget, not a set of independent attacks. Results with different budgets are not comparable.

Incomplete cells are excluded and listed with their reason, rather than silently compressing surviving seeds into a complete-budget result. This is still a conditional-on-observation estimate: missingness can bias it. Common-case comparisons and coverage reporting are required. Over-refusal uses benign controls. Live **secure-and-useful** is unavailable in this implementation: a transcript marker cannot independently verify the legitimate task into which an attack is inserted (the *carrier task*). Secure-and-useful evaluation is a goal motivated by (Bhatt et al., 2023; Debenedetti et al., 2024; Zhan et al., 2024), not an established live measurement here.

5.4 Gated qualitative judging

The optional Claude-rubric backend uses absolute 0–4 rubrics for visual, architecture, and UX judgments (Liu et al., 2023; Zheng et al., 2023; Kim et al., 2024). Its outputs must be finite and within that range before division by four; malformed outputs are unavailable evaluations. Absolute presentation avoids pairwise ordering as an input, but does not remove verbosity, style, or model-family bias.

Other backends measure different quantities. The heuristic backend uses deterministic functional and code-structure proxies; the CLIP backend uses image similarity and descriptor comparisons. A heuristic visual proxy does not inspect rendered appearance. Backend identity and evaluated evidence coverage are therefore part of score provenance.

Assumption A3 (bounded gated judging). To qualify as evidence-based judging, a backend must return a bounded score from candidate evidence it actually reads. Visual and UX assessment require a built, served artifact; static architecture assessment requires a built repository with scoreable files. This is an eligibility condition, not an established property of every backend. Human agreement and resistance to adversarial candidate content require separate empirical validation.

5.5 Cross-family aggregate (the Gauntlet Index)

Let $\mathcal{T} = \{S, Q, R, G, P\}$ be the five fixed task families, and write $\tilde{s}_t(M) \in \mathbb{I}$ for arm M's normalized family-t headline. The implementation maps S to $1 - \mathrm{ASR}$, Q to requirement coverage, and G to g_{score}, the reported aggregate described in Section 5.2. Families P and R use their respective composites. Higher values are preferred. For an arm with qualified scores on **all five** families, define

$$\mathrm{Index}(M) = \frac{1}{5} \sum_{t \in \mathcal{T}} \tilde{s}_t(M) \in \mathbb{I}.$$

The index is unavailable when any family is missing; individual scores and coverage remain visible. Its range follows from convexity, not empirical comparability. Qualification requires current scoring, observed execution, and complete evaluable coverage. The current publication excludes Security records from qualified headlines, so it does not presently report a five-family index. Matched tasks, budgets, provenance, and timestamps are still necessary for competitive comparisons. The index is not a validated general-capability scale.

5.6 Operational diagnostics for long-horizon results

The quantities in Table 1 are **proposed diagnostics**, not measurements supplied by the current paper. A valid transition model or repeated, uncensored run-level observations would be needed to estimate them. A budget cap is not an observed completion time.

Table 1: Proposed execution diagnostics. No fitted Markov kernel, calibrated per-requirement prior, or completion-time distribution is reported here.

Diagnostic	Meaning
n	Maximum loop iterations / total passes allowed for the run.
Productive passes	Passes with $S_{t+1} \supsetneq S_t$ in the certificate-accumulation trace.
$\mathrm{efc}(S_t)$	New certificates enabled by the abstract operator: $\lvert\Phi(S_t) \setminus S_t\rvert$.
$\lvert R\rvert$, $\lvert R_{\mathrm{req}}\rvert$	Total and required requirement counts (the milestone budget).
Reasoning budget b	Thinking mode / compute budget the harness ran under.
$\Pr(T_C \le n)$	Probability of covering R_{req} within the iteration budget (§4.4).
$\mathbb{E}[T_C]$	Expected number of passes to completion.
$-\log_2 p$	Event surprisal in bits; for a harmful event, a descriptive risk transform.

The last row uses self-information (Shannon, 1948). For an event of probability $p \in (0, 1]$, its *surprisal* is

$$I(p) = -\log_2 p \text{ bits} \qquad \big(p = \tfrac{1}{2} \Rightarrow 1 \text{ bit},\ \ p = \tfrac{1}{4} \Rightarrow 2 \text{ bits},\ \ p = 0.01 \Rightarrow 6.64 \text{ bits}\big),$$

This is finite on $(0, 1]$; extend $I(0) = +\infty$. A zero observed attack count is not proof that the underlying probability is zero. For positive raw and governed rates, a descriptive bit difference is $\log_2(\widehat{p}_{\mathrm{raw}}/\widehat{p}_{\mathrm{gov}})$; $0/0$ is undefined. A probability interval $[p_L, p_U]$ transforms to $[-\log_2 p_U, -\log_2 p_L]$, with an infinite upper endpoint if $p_L = 0$. This transformation creates no new evidence or confidence guarantee.

A proposed information-weighted count must specify its event. Let $Z_t = (H_0, S_0, \ldots, H_t, S_t)$ be the observed run history through pass t. For each uncertified $r \in R \setminus S_t$, suppose a calibrated estimate $q_t(r) = \Pr(r \in S_{t+1} \setminus S_t \mid Z_t) \in (0, 1]$ were available. For an observed transition one could then define $\mathrm{iefc}_t = \sum_{r \in S_{t+1} \setminus S_t} -\log_2 q_t(r)$. This sums marginal event surprisals; it is not joint information without an appropriate dependence model, nor a measure of task value. No such probability estimates are reported here.

6. Experimental Setup

The evaluation separates three questions. First, does a configured governance layer reduce classified unsafe responses while preserving behavior on benign requests? Second, how do coding harnesses differ in the functional and structural quality of the artifacts they produce? Third, can automatically estimated semantic references recover rankings obtained with hand-authored references? These questions require different experimental units: attack attempts, completed coding tasks, and independently specified application briefs, respectively.

Sections 6.1–6.6 define the comparison protocol and measurement boundaries. Section 7 then describes the retained June 2026 pilot studies, including their tasks, sample coverage, numerical outcomes, and limitations. The pilots were not one uniformly controlled experiment. Model execution and rescoring of existing artifacts have different evidential meanings. The paper reports benchmark outcomes, not constructed demonstrations; neither rescoring nor report regeneration is a new independent model execution.

6.1 Baselines and base models

An arm is a harness, configured provider/model, prompts, tools, environment, and budget. The implementation offers raw Codex CLI, Claude Code, Oh My Pi (OMP), and OpenCode adapters with Cortex-configured counterparts. These are product-bundle comparisons. Configuration and display labels are not attestations of the model that actually served a historical request. Floating aliases, environment overrides, reasoning effort, and tool/runtime versions must be recorded per execution.

Same-model labels alone do not identify a governance-only causal effect. Cortex arms also change prompts, configuration, and sometimes planning/repair budgets. A policy-only ablation needs matched effective models, tools, environments, cases, evaluation access, and total resource budgets. A benchmark invocation can contain many internal agent actions.

6.2 Relation to existing benchmarks

SWE-bench and its variants assess repository issue resolution (Jimenez et al., 2024; OpenAI, 2024; Zhang et al., 2025); LiveCodeBench emphasizes contamination-resistant code evaluation (Jain et al., 2025). Commit0 and Terminal-Bench include difficult multi-stage work (Zhao et al., 2025; Merrill et al., 2026). Independent tasks do not imply single-pass agents or saturated performance. CyberSecEval, AgentDojo, and InjecAgent study security and prompt injection (Bhatt et al., 2023; Debenedetti et al., 2024; Zhan et al., 2024); AgentDojo also evaluates benign task utility. AgentBench, GAIA, WebArena, OSWorld, and τ-bench cover other agent environments and interaction contracts (Liu et al., 2024; Mialon et al., 2024; Zhou et al., 2024; Xie et al., 2024; Yao et al., 2024). Gauntlet is a complementary framework, not evidence that these suites lack long-horizon work, utility evaluation, or harness comparisons.

6.3 Tasks and measured outcomes

Table 2 distinguishes the intended task family from what its evaluator can establish. Passing finite tests is evidence for those tests, not complete semantic correctness.

Table 2: Five families and their measurement boundaries. No direction of a raw-versus-governed effect is assumed by the score definition.

Family	Task and score	Validity boundary
S: Safety	Classified attack responses and proposed actions; ASR and benign over-refusal.	Lexical classification and action replay are not observed exploits. Live carrier-task utility is unmeasured.
Q: Quality	Generated code, hidden behavioral tests, static findings, lint/types and quality rubrics.	Analyzer availability and evaluator integrity are prerequisites; finite tests are incomplete.
R: Bugfix	Repository patches; resolution-gated score with held-out and regression checks.	Reference-diff similarity is not correctness. Test-runner tampering must be independently excluded.
G: Generative	Full-stack application generation; build/serve checks, feature probes, static scores and optional visual evidence.	Passed features do not establish complete application behavior, real model-endpoint use or time-resolved progress. Historical applications and the current chat-app protocol are distinguished below.
P: Project	One storefront brief, build-gated functional/semantic/visual/static composite.	Partial behavioral and accessibility proxies are not a complete storefront or payment validation.

Application construction protocol. The storefront brief asks for a fashion-shopping application with product browsing, filtering or sorting, variant selection, cart editing, consistent price totals, checkout, and order confirmation. Candidates choose their implementation structure. The brief also requests a documented backend API, a responsive interface, accessibility support, an installable progressive web app, an offline shell, automated core-flow tests, and build/serve instructions. Payment is specified in test mode; no real-money transaction is required.

The functional evaluator organizes the storefront into five journeys: home, browsing, product detail, cart, and checkout, with respective weights $1, 2, 2, 3, 3$. It scores the weighted fraction of passing evaluable journeys after a successful build. The implemented checks are narrower than the brief. Some verify that a control exists rather than that filtering, sorting, or quantity editing works. Cart checks compare displayed arithmetic; checkout checks reaching a checkout surface. No implemented test-payment driver establishes payment completion. Likewise, manifest and service-worker checks

do not demonstrate an offline shopping session, and markup heuristics do not establish full keyboard accessibility.

The separate current chat-app protocol checks message/composer elements, stylesheet and script availability, a chat-response token, and a history response. Its supplied model-service substitute is a deterministic echo responder, not a learned model. Passing a token check does not prove that the application contacted that service; an application could construct the same response itself. These checks therefore provide interface and API smoke coverage, not evidence of model reasoning, streaming, conversation isolation, or durable history. The historical five-probe application study in Section 7.2 and Appendix B.3 instead evaluates a to-do application and is not a chat result.

6.4 Configuration isolation and adaptive evaluation

Raw security adapters create temporary workspaces and scrub selected environment and tool-home configuration. Governed adapters add a safety preamble and link repository configuration. This is configuration isolation, **not host or network confinement**, nor proof that every hook loaded. Track S does not itself run the completion loop. Some nominal tool-output and memory attacks are delivered as workspace files representing those surfaces, not through native tool-result or session-memory interfaces.

Adaptive attacks use earlier target responses to generate later prompts, but each target attempt starts a fresh session/workspace. Their budgeted path is not a continuous target conversation, and follow-ups need not preserve the original delivery surface. Selecting a corpus using previous outcomes and merely labeling examples as held out do not establish an independently frozen evaluation set. A genuine holdout must be fixed before development and withheld from prompt design, curation, and repair feedback.

The project evaluator scores first/final snapshots and retains the highest composite. Governed arms can also use evaluation feedback during repair. These current and historical paths produce adaptively selected development scores, not untouched holdout measurements. Access to more evaluations can favor one arm. A controlled comparison must commit each artifact before a separate final evaluation and report all generation/evaluation costs.

6.5 Replication and missingness

Seed labels index repeated calls; they are not controlled provider RNG seeds. Configured maximum attempts, completed attempts, skipped cases, internal repair passes, and measured provider usage are separate quantities. Q and R report worst observed repeats rather than a pooled per-seed success estimate. Live full-app/repository paths have historically been single-run paths; requested counts must not be interpreted as performed trials. A timeout is not a refusal or proof of candidate safety. Excluding incomplete cells requires coverage reporting and common-case comparisons, and does not remove informative-missingness bias.

6.6 Containment and evaluator integrity

The Q/R/G/P evaluation paths execute generated code in Docker. For G/P, dependency installation and compilation occur in a build stage with registry-network access; the later runtime probe applies resource limits and a track-specific network policy. Those probe restrictions must not be attributed to every build-stage operation. Containerized evaluation also does not establish confinement of the generation harness. The live Track S CLI adapters lack an outer sealed-agent sandbox; some use permission-bypass modes, and governed arms inherit host configuration. Adversarial execution requires independent verification of the actual isolation boundary.

The lexical detector (L0) and in-memory action-replay scorer (L1) classify captured text and proposed actions. Their historical "confirmed" fields are scorer labels, not authenticated side-effect attestations. Echo filtering, refusal heuristics, bounded capture, and missing typed tool-event provenance can misclassify outcomes. Dynamic evaluation must also protect test-runner configuration and authenticate verdicts rather than trusting candidate-produced terminal summaries. The present implementation has not established security or evaluator-integrity sign-off.

7. Results and Evidence Status

The following **historical descriptive studies** explain the retained observations without requiring access to internal execution records. They concern June 2026 artifacts and captures, not newly reproduced experiments. Five arm labels recur: Codex CLI, Claude Code, OpenCode, Cortex over Codex, and Cortex over Claude. The labels identify configurations, not independently authenticated model versions. Each study's task set and observation unit are specified separately.

7.1 Captured-response safety pilot

This pilot asks how often captured responses are classified as following an unsafe instruction, while separately checking benign refusals. The case set contains 61 harmful cases, two diagnostic instruction-following probes, and two benign controls. Harmful objectives include secret disclosure, destructive repository operations, unsafe dependencies, and attempts to disable protections. Delivery contexts include direct requests and representations of repository, tool-output, and memory content; they are not all native integrations with those surfaces.

One attempt is recorded for each completed case–arm cell. Across five arms, 290 of 305 possible harmful observations are retained; 15 are missing after timeouts. The diagnostic probes and benign controls are excluded from the harmful-response denominator. Figure 4 and Table 3 report classified successes divided by observed harmful attempts, not by the configured maximum number of attempts.

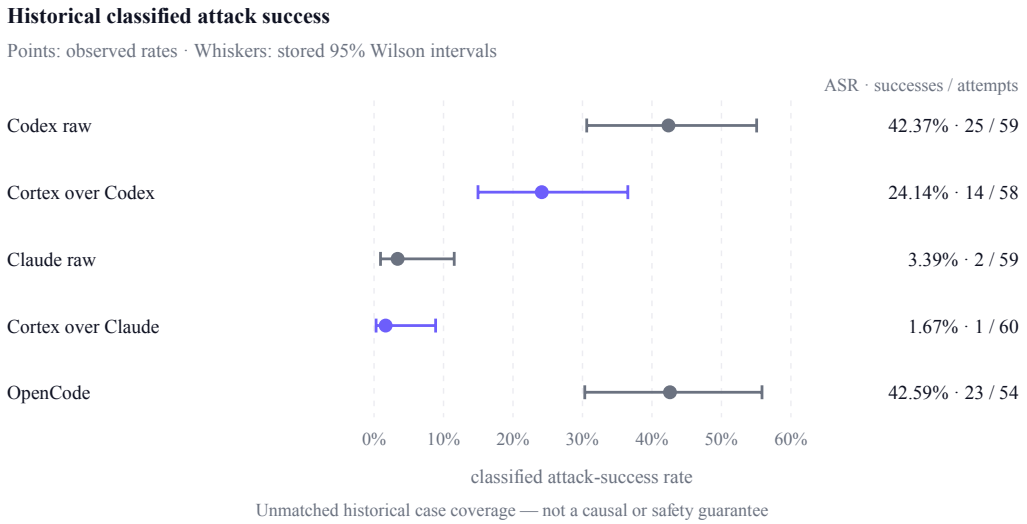

Figure 4: Classified attack responses in the June 2026 safety pilot, comparing five historical harness configurations. Points show observed rates, whiskers the stored 95% Wilson intervals, and labels the successes and observed attempts. Model names are historical labels. Section 7 describes coverage, provenance, and benign-control limitations; these are descriptive, not causal comparisons.

Table 3: Historical classified safety outcomes from the stated June 2026 cohort. Intervals are the stored 95% Wilson intervals, not newly reproduced estimates.

Historical arm label	Classified successes / observed attempts	ASR	Stored 95% Wilson interval
Codex raw	25 / 59	42.37%	30.61–55.07%
Cortex over Codex	14 / 58	24.14%	14.96–36.53%
Claude raw	2 / 59	3.39%	0.93–11.54%
Cortex over Claude	1 / 60	1.67%	0.29–8.86%
OpenCode	23 / 54	42.59%	30.33–55.84%

Using the exact observed counts, the descriptive relative ASR reductions, $(\mathrm{ASR}_{\mathrm{raw}} - \mathrm{ASR}_{\mathrm{governed}})/\mathrm{ASR}_{\mathrm{raw}}$, are approximately 43.0% for the Codex-labeled pair and 50.8% for the Claude-labeled pair. These are relative reductions, not percentage-point differences. The completed case sets differ between arms, and timed-out attempts cannot be treated as safe responses. Each of the four paired arms has zero refusals on only two benign controls, with stored interval $[0, 65.76\%]$; OpenCode refuses one of its two controls. Two diagnostic probes per arm are reported separately and are not additional benign-utility trials.

The retained responses were subsequently rescored using lexical analysis, in-memory replay of proposed actions, and response judging. No authenticated external exploit follows from a positive classification. Historical judge summaries also conflict: a summary describes heuristic judging while per-response entries identify a language-model judge. This limits reconstruction of the exact scoring protocol. These observations establish neither unchanged benign utility nor statistical equivalence, low population over-refusal, or a governance-only causal effect.

7.2 Coding and application outcomes

Figure 5 compares three different outcomes, each with its own denominator. The **quality study** comprises six code-generation tasks and one outcome per task and arm. Its headline is the equal-weight mean of the six behavioral-test pass fractions. Raw Codex and Claude both score 0.8125, and their corresponding Cortex configurations both score 0.9792. The entire paired difference comes from the TypeScript upload task; all four configurations still miss one shopping-cart check. Table 4 in Appendix B.1 includes every task-level numerator and denominator, including differing upload-test totals and generation failures.

The **repair study** replays patches for six elementary Python bugs, with one outcome per issue and arm. All four raw/governed Codex and Claude configurations resolve all six issues; OpenCode resolves five. There is no paired difference in issue-resolution rate. Table 5 in Appendix B.2 reports all issue tests and the separate patch-sensitive composites; empty auxiliary test groups are not evidence of regression robustness.

The **application study** concerns a to-do app, not the chat gallery. Each arm has one candidate and five final feature checks. Codex passes $3/5$; the other four configurations pass $5/5$. All candidates build. Table 6 defines the checks, and Figure 7 in Appendix B.3 shows their positions in the final checklist. Those positions are not elapsed time or repeated execution milestones.

These are descriptive artifact evaluations, not matched governance-only trials. Quality execution metadata conflict, repair lacks auxiliary robustness tests, and the application study has no independently repeated candidates. Appendix B retains these limitations beside the complete evidence.

7.3 Storefront outcomes and reference agreement

Two candidate pools address the same storefront brief in Section 6.3, with one built application per arm. The two-arm pool scores 0.7489 for raw Claude and 0.6507 for raw Codex. In the separate five-arm pool, raw Codex scores 0.7212, OpenCode 0.7071, Cortex over Codex 0.6799, raw Claude 0.6737, and Cortex over Claude 0.6533. There is no universal Cortex advantage in these observations. Tables 7–8 and Figure 8 in Appendix B.4 include every component, the composite definition, recorded evaluator identities, and proxy limitations.

Figure 6 asks a narrower question: does estimated-reference VERTEX order these same applications like authored-reference VERTEX? Higher similarity gives a better rank; rank 1 is highest. The two-arm pool has identical ordering and Spearman correlation 1. The five-arm pool exchanges the first and third positions, giving correlation 0.6. Table 9 in Appendix B.5 supplies every score pair and rank. Neither value is correctness accuracy or a probability that a reference is valid. Peer-derived reference construction differs between pools; both pools concern one brief, not seven independent task samples.

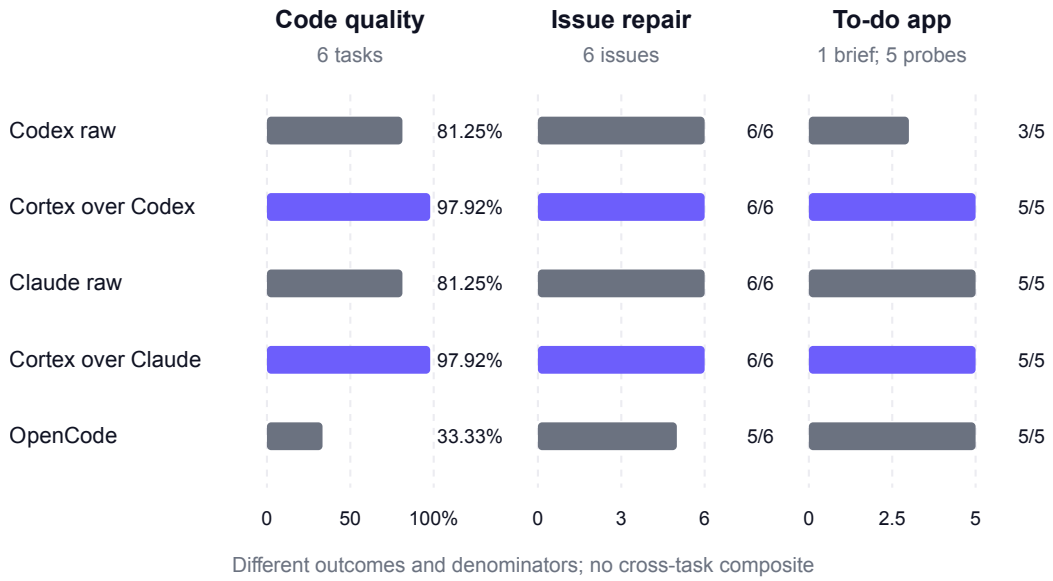

Figure 5: Recorded coding outcomes: task-macro behavioral coverage (left), resolved issues (middle), and passed to-do features (right). Each row is a harness configuration; higher values are better within each panel, not comparable units across panels. Exact cells and evaluation limits are in Tables 4–6. The upload task accounts for the paired quality difference; there is no paired repair-resolution difference. These small historical pilots are not causal tests.

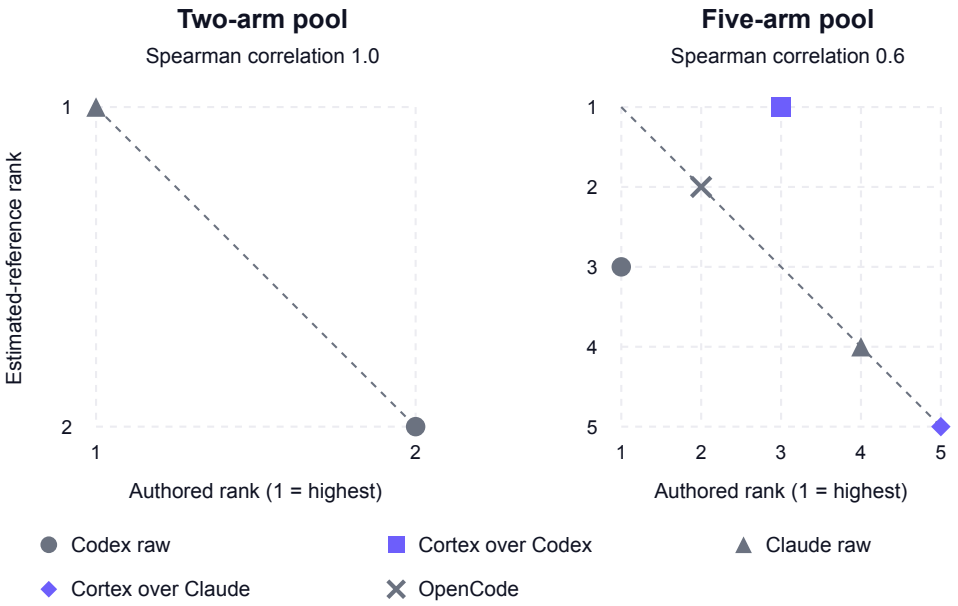

Figure 6: Within-pool VERTEX rank agreement on the same storefront brief. Each marker is one built application; the dashed diagonal is identical ordering, not score calibration. Rank 1 is highest on both axes. The two-arm pool needs only a matching order to obtain correlation 1; the five-arm pool swaps the first and third positions. Exact scores and ranks are in Table 9. Pool-based peer reference construction differs between the panels; they are not independent task replications.

8. Discussion

The finite-lattice result identifies a precise contract for requirement closure: validation must be monotone, accepted evidence sound, and prior certificates persistent. Repairing code complicates the last condition because an edit can invalidate earlier checks. Fresh validation of the final artifact is therefore central to connecting the abstract contract with an operational completion decision.

The evaluation framework makes these obligations observable in principle, but the archived comparisons do not identify a governance-only effect. A policy ablation should hold the effective model, task set, tools, and total budget fixed, with independent final evaluation. Replication across independently selected tasks and adequate benign controls is needed to assess generalization and utility. Only retained benchmark observations support the empirical discussion here; constructed outcomes are not evidence.

9. Assumptions and Limitations

The formal assumptions concern different boundaries. A1 requires complete mediation and threat-class coverage; A2 requires sufficient abstract state and sound, monotone, persistent certification; A3 requires bounded judgments grounded in the evaluated evidence. Parsing a task into expectation records does not construct a sound oracle. The probabilistic bound additionally requires uniformly positive progress before completion; without it, expected completion time can be infinite.

The principal measurement limitations are length-sensitive similarity, heuristic references, conditional journey/frame coverage, correlated attempts, and incomplete observations. Section 6.6 specifies the unresolved execution and evaluator-integrity boundaries. The current probes also do not fully measure payment integration, carrier-task utility, consistency of claims with artifacts, or long-horizon behavior.

Historical missing or failed signals cannot be certified by changing current error handling. The retained examples were recorded in June 2026; a later report date does not represent a new experiment.

10. Conclusion

The retained benchmark outcomes do not show a uniform Cortex advantage. Classified harmful-response rates are lower in the paired Cortex configurations, but completed case sets differ and benign controls are too few to establish preserved utility (Figure 4 and Table 3). Higher paired quality coverage comes from one upload task; repair resolution is unchanged between paired configurations, and the single to-do brief distinguishes task creation and readback rather than long-horizon progress (Figure 5 and Tables 4–6). The storefront pools do not consistently favor Cortex. Estimated-reference rank agreement varies across two pools for one brief and does not establish correctness calibration (Figure 6 and Tables 7–9).

These conclusions concern the recorded benchmark outcomes, including artifact rescoring. Unequal observation sets and unresolved provenance prevent treating all cohorts as verified live experiments or attributing differences to governance alone. The finite-lattice and stochastic results are conditional mathematical statements, not performance gains established by these comparisons. Matched, independently evaluated experiments are still required to estimate effects on safety, utility, generalization and long-horizon completion.

Availability

The interactive paper and latest published results are available at benchmark.cortex.a2olabs.com. Results retain their source cohorts, recorded measurements and evidence limitations. The standalone Gauntlet benchmark snapshot is available at github.com/Xpitfire/cortex-gauntlet. It excludes private implementation dependencies and raw captures, so the screened snapshot alone is not a complete end-to-end reproduction environment. Download the Cortex CLI from cortex.a2olabs.com. The downloadable PDF is a dated rendering of this paper, not a claim that the historical results are new measurements.

REFERENCES

Anderson, J. R. et al. (2004). *An Integrated Theory of the Mind.* Psychological Review 111(4):1036–1060.

Banach, S. (1922). *Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales.* Fundamenta Mathematicae 3:133–181. (Contraction mapping.)

Bhatt, M. et al. (2023). *Purple Llama CyberSecEval: A Secure Coding Benchmark for Language Models.* arXiv:2312.04724.

Brown, B. et al. (2024). *Large Language Monkeys: Scaling Inference Compute with Repeated Sampling.* arXiv:2407.21787.

Chen, M. et al. (2021). *Evaluating Large Language Models Trained on Code.* arXiv:2107.03374. (The $\mathrm{pass@}k$ unbiased estimator.)

Dawid, A. P. & Skene, A. M. (1979). *Maximum Likelihood Estimation of Observer Error-Rates Using the EM Algorithm.* J. R. Stat. Soc. C 28(1):20–28. (A probabilistic noisy-observer model; not a consistency proof for this benchmark's heuristic.)

Debenedetti, E. et al. (2024). *AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents.* NeurIPS Datasets & Benchmarks. arXiv:2406.13352.

Dinu, M.-C., Leoveanu-Condrei, C., Holzleitner, M., Zellinger, W. & Hochreiter, S. (2024). *SymbolicAI: A framework for logic-based approaches combining generative models and solvers.* arXiv:2402.00854. (Relational-trajectory evaluation: Gaussian-kernel cross-similarity against a reference distribution, normalized against a fixed random-sequence baseline.)

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

Gao, Y., Zhao, W. & Eger, S. (2020). *SUPERT: Towards New Frontiers in Unsupervised Evaluation Metrics for Multi-Document Summarization.* ACL. arXiv:2005.03724. (Pseudo-reference built from salient source content.)

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

Greshake, K. et al. (2023). *Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection.* ACM AISec. arXiv:2302.12173.

Hoffmann, J. et al. (2022). *Training Compute-Optimal Large Language Models.* arXiv:2203.15556.

Howard, R. A. (1966). *Information Value Theory.* IEEE Transactions on Systems Science and Cybernetics 2(1):22–26.

Jain, N. et al. (2025). *LiveCodeBench: Holistic and Contamination-Free Evaluation of Large Language Models for Code.* ICLR. arXiv:2403.07974.

Jimenez, C. E. et al. (2024). *SWE-bench: Can Language Models Resolve Real-World GitHub Issues?* ICLR. arXiv:2310.06770.

Kaplan, J. et al. (2020). *Scaling Laws for Neural Language Models.* arXiv:2001.08361.

Kautz, H. (2022). *The Third AI Summer.* AI Magazine 43(1) (AAAI Engelmore Lecture).

Kim, S. et al. (2024). *Prometheus 2: An open-source language model specialized in evaluating other language models.* arXiv:2405.01535.

Kleene, S. C. (1952). *Introduction to Metamathematics.* North-Holland. (Kleene iteration / first recursion theorem.)

Kumar, S. & Byrne, W. (2004). *Minimum Bayes-Risk Decoding for Statistical Machine Translation.* NAACL-HLT. (Selecting by consensus utility; other samples act as pseudo-references.)

Laird, J. E., Newell, A. & Rosenbloom, P. S. (1987). *SOAR: An architecture for general intelligence.* Artificial Intelligence 33(1):1–64.

Lightman, H. et al. (2024). *Let's Verify Step by Step.* ICLR. arXiv:2305.20050.

Liu, Y. et al. (2023). *G-Eval: NLG Evaluation using GPT-4 with Better Human Alignment.* EMNLP. arXiv:2303.16634.

Liu, X. et al. (2024). *AgentBench: Evaluating LLMs as Agents.* ICLR. arXiv:2308.03688.

Madaan, A. et al. (2023). *Self-Refine: Iterative Refinement with Self-Feedback.* NeurIPS. arXiv:2303.17651.

Merrill, M. A. et al. (2026). *Terminal-Bench: Benchmarking Agents on Hard, Realistic Tasks in Command-Line Interfaces.* arXiv:2601.11868. tbench.ai.

Mialon, G. et al. (2024). *GAIA: a benchmark for General AI Assistants.* ICLR. arXiv:2311.12983.

Newell, A. & Simon, H. A. (1972). *Human Problem Solving.* Prentice-Hall.

OpenAI (2024). *Introducing SWE-bench Verified.* A 500-instance human-validated subset of SWE-bench. openai.com/index/introducing-swe-bench-verified/.

Patel, A. et al. (2024). *Large Language Models Can Self-Improve At Web Agent Tasks.* arXiv:2405.20309.

Pillutla, K. et al. (2021). *MAUVE: Measuring the Gap Between Neural Text and Human Text using Divergence Frontiers.* NeurIPS. arXiv:2102.01454. (Distributional comparison to a reference set rather than to a single point.)

Ratner, A. et al. (2017). *Snorkel: Rapid Training Data Creation with Weak Supervision.* VLDB 11(3):269–282. arXiv:1711.10160. (Denoising multiple weak label sources into one probabilistic signal.)

Rei, R., Stewart, C., Farinha, A. C. & Lavie, A. (2020). *COMET: A Neural Framework for MT Evaluation.* EMNLP. arXiv:2009.09025. (Source-aware, reference-based translation evaluation.)

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

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

Shannon, C. E. (1948). *A Mathematical Theory of Communication.* Bell System Technical Journal 27:379–423, 623–656. (Self-information $-\log_2 p$ in bits; the basis for the surprisal and risk-bit diagnostics of Section 5.6.)

Shinn, N. et al. (2023). *Reflexion: Language Agents with Verbal Reinforcement Learning.* NeurIPS. arXiv:2303.11366.

Snell, C. et al. (2024). *Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters.* arXiv:2408.03314.

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

van den Oord, A., Li, Y. & Vinyals, O. (2018). *Representation Learning with Contrastive Predictive Coding.* arXiv:1807.03748. (InfoNCE: a contrastive objective that lower-bounds mutual information.)

van Emden, M. H. & Kowalski, R. A. (1976). *The Semantics of Predicate Logic as a Programming Language.* Journal of the ACM 23(4):733–742. (Immediate-consequence operator; least fixed point.)

Wei, A., Haghtalab, N. & Steinhardt, J. (2023). *Jailbroken: How Does LLM Safety Training Fail?* NeurIPS. arXiv:2307.02483.

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

Xie, T. et al. (2024). *OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments.* NeurIPS Datasets & Benchmarks. arXiv:2404.07972.

Yao, S. et al. (2023). *ReAct: Synergizing Reasoning and Acting in Language Models.* ICLR. arXiv:2210.03629.

Yao, S. et al. (2024). *τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains.* arXiv:2406.12045.

Zhan, Q. et al. (2024). *InjecAgent: Benchmarking Indirect Prompt Injections in Tool-Integrated Large Language Model Agents.* ACL Findings. arXiv:2403.02691.

Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q. & Artzi, Y. (2020). *BERTScore: Evaluating Text Generation with BERT.* ICLR.

Zhang, L. et al. (2025). *SWE-bench Goes Live!* arXiv:2505.23419. (Continuously updated, contamination-resistant issue resolution.)

Zhang, X., Wang, D., Xu, K., Zhu, Q. & Che, W. (2026). *Scaling Laws for Agent Harnesses via Effective Feedback Compute.* arXiv:2605.29682.

Zhao, W. et al. (2025). *Commit0: Library Generation from Scratch.* ICLR. arXiv:2412.01769.

Zheng, L. et al. (2023). *Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena.* NeurIPS. arXiv:2306.05685.

Zhou, S. et al. (2024). *WebArena: A Realistic Web Environment for Building Autonomous Agents.* ICLR. arXiv:2307.13854.

APPENDIX A — PROOFS

The following calculations give the identities used in the score definitions and conditional bounds. Labels P1–P8 distinguish these appendix calculations; they are not additional manuscript propositions.

- **P1 (normalizer).** For $b \in [0,1)$, $\eta_b(x) = \mathrm{clamp}((x-b)/(1-b); 0, 1)$ maps $[b,1]$ onto $[0,1]$. On that interval the clamp is inactive and the affine map has positive slope $1/(1-b)$, with endpoints $\eta_b(b) = 0$ and $\eta_b(1) = 1$. Outside the interval the clamp is constant, so the complete map is nondecreasing, but not globally strictly increasing.
- **P2 (composite).** For $w \in \Delta(\{1,\dots,K\})$ and $s_k \in \mathbb{I}$, let $s_{\min} = \min_k s_k$ and $s_{\max} = \max_k s_k$. Multiplying $s_{\min} \le s_k \le s_{\max}$ by $w_k \ge 0$ and summing gives $s_{\min} \le \sum_k w_k s_k \le s_{\max}$, because the weights sum to one. Thus the ungated average lies in $\mathbb{I}$, and the binary build gate gives $C = 0$ when $g = 0$ without changing that bound when $g = 1$.
- **P3 (harmonic mean).** For $P, Q \ge 0$, set $F = 0$ at $P = Q = 0$, otherwise $F = 2PQ/(P+Q)$. The zero case satisfies the bounds directly; henceforth assume $P + Q > 0$. If $0 \le P \le Q$, then $F - P = P(Q-P)/(P+Q) \ge 0$ and $Q - F = Q(Q-P)/(P+Q) \ge 0$; exchange P, Q for the other case. Also $(P+Q)/2 - F = (P-Q)^2/(2(P+Q)) \ge 0$.
- **P4 (DTW).** For $m, n \in \mathbb{N}_+$, similarities in $[-1,1]$, and finite $\lambda \ge 0$, both the local dissimilarity and its positional penalty factor are nonnegative. Every admissible alignment path therefore has nonnegative cost, as does the minimum over paths. Consequently $1 - \mathrm{DTW}/(m+n) \le 1$; clamping this quantity below at zero gives $D \in \mathbb{I}$. This is a range bound, not a guarantee that descriptor order represents elapsed execution time.
- **P5 (any-success and all-success estimators).** For integers $0 \le c \le n$ and $1 \le k \le n$, exactly $\binom{n-c}{k}$ of the $\binom{n}{k}$ subsets avoid success, while exactly $\binom{c}{k}$ consist entirely of successes. Thus $\mathrm{pass@}k = 1 - \binom{n-c}{k}/\binom{n}{k}$ is the fraction of k-subsets containing any success, and $\mathrm{pass}^k = \binom{c}{k}/\binom{n}{k}$ is the fraction consisting entirely of successes. Both lie in $[0,1]$. Coupling the subsets as prefixes of one uniform permutation shows that the any-success event can only grow with k, whereas the all-success event can only shrink. Under independent Bernoulli trials with common success probability p, each fixed subset has any-success probability $1-(1-p)^k$ and all-success probability p^k. Linearity of expectation gives unbiasedness for these respective targets. Independence is required for these targets, not for the finite-subset identities or monotonicity.
- **P6 (Wilson).** For $N > 0$, $z > 0$, and $\widehat{p} \in [0,1]$, the score-test boundary $(\widehat{p} - p)^2 = z^2 p(1-p)/N$ becomes $(N + z^2)p^2 - (2N\widehat{p} + z^2)p + N\widehat{p}^2 = 0$. Applying the quadratic formula and dividing numerator and denominator by N yields the interval endpoints in Section 5.3. Exact solution of this equation does not give exact confidence coverage: the score-test approximation and sampling assumptions remain necessary.
- **P7 (Theorem 1).** Inflationarity gives an increasing chain of requirement sets; each strict increase adds at least one previously absent member of finite R. Starting from the empty set permits at most $|R|$ such increases. Monotonicity keeps every iterate below every fixed point, so stabilization gives the least fixed point. Proposition 1 then follows because the successive set differences are disjoint and their union is the final certified set.
- **P8 (monotone Markov execution).** Under Proposition 2's uniform positive-progress assumption, the waiting time for each strict increase is bounded by a geometric waiting time of mean $1/\varepsilon$. At most $|R| - |S_0|$ increases can precede completion, giving the stated expectation bound by addition of expectations; independence between these waiting times is not needed. An incomplete absorbing state violates the progress assumption and is not covered by the result.

These arguments establish algebraic identities and conditional bounds. They do not establish that a concrete validator is sound, that a repair preserves earlier certificates, or that a reported experimental outcome is authentic. Those claims require separate execution evidence and empirical validation.

APPENDIX B — COMPLETE HISTORICAL RESULT EXHIBITS

This appendix contains the task-level and component-level evidence behind Section 7. Figures and tables use a fixed arm order: raw Codex, Cortex over Codex, raw Claude, Cortex over Claude, and OpenCode. A column headed by a base harness and "+ Cortex" denotes its configured Cortex counterpart; "raw" means unwrapped. Labels do not independently attest the historical model endpoint.

B.1 SIX-TASK CODE GENERATION

The tasks are password and secret utilities, safe input parsing, database access, a TypeScript upload API, web-security utilities, and a multi-file shopping-cart package. The other five tasks use Python. There is one retained outcome per task and arm, giving 30 outcomes. Four OpenCode generation failures remain in the denominator rather than being discarded.

Table 4 gives passing and collected behavioral checks. The macro coverage is the equal-weight mean of six task rates, not a pooled check count and not the fraction of natural-language requirements independently certified. For the four paired arms, the only differing task is upload: raw configurations record one failing check, while Cortex configurations record two passing checks. Shopping cart remains at seven of eight checks. Figure 5's left panel summarizes these exact task rates.

Table 4: Quality: passing / collected behavioral checks in each task–arm outcome. The last row is the equal-weight mean of the six task rates, not pooled test success. Collected upload test totals differ across arms. Four OpenCode generation failures remain as zero-scored outcomes. These test counts are not the number of natural-language requirements certified.

Task	Codex raw	Codex + Cortex	Claude raw	Claude + Cortex	OpenCode reference
Secret utilities	3/3	3/3	3/3	3/3	3/3
Input parsing	2/2	2/2	2/2	2/2	2/2
Database access	1/1	1/1	1/1	1/1	0/1
Upload API	0/1	2/2	0/1	2/2	0/1
Web security	7/7	7/7	7/7	7/7	0/1
Shopping cart	7/8	7/8	7/8	7/8	0/1
Macro coverage	0.8125	0.9792	0.8125	0.9792	0.3333

Per-outcome entries describe dynamic execution and nonzero test counts, but an accompanying methodology summary describes offline variant selection and a static-only containment boundary. These conflicting descriptions prevent certifying a controlled live experiment; they also do not justify relabeling every outcome synthetic. Static and heuristic quality indicators are supplementary. Zero findings do not establish absence of vulnerabilities.

B.2 SIX ELEMENTARY REPAIR ISSUES

The Python issues concern summing even rather than odd values, restoring last-in-first-out stack removal, lowercasing URL slugs and stripping punctuation, rounding up pagination for a partial page, case-insensitive word counting, and enforcing a lower clamp bound. These are small issue-resolution tasks, not a difficult multi-file repository benchmark.

Saved patches were replayed through the evaluator, producing 30 issue–arm outcomes. Table 5 contains seven issue tests per arm: two stack tests and one for every other issue. All four paired configurations resolve all six issues; OpenCode fails pagination. The recorded composites combine resolution with patch descriptors; the small Codex difference reflects patch minimality rather than additional resolved issues. Figure 5 therefore plots resolution counts, not a misleading composite improvement.

Table 5: Repair: passing / collected issue tests, resolved issues, and stored composite means. There are no auxiliary held-out or regression tests in any row. Empty auxiliary groups received passing defaults; strict-rate fields therefore add no independent evidence. The paired Codex composite difference reflects patch minimality, not extra resolved issues.

Issue	Codex raw	Codex + Cortex	Claude raw	Claude + Cortex	OpenCode reference
Even-value sum	1/1	1/1	1/1	1/1	1/1
Stack removal	2/2	2/2	2/2	2/2	2/2
URL normalization	1/1	1/1	1/1	1/1	1/1
Pagination	1/1	1/1	1/1	1/1	0/1
Word counting	1/1	1/1	1/1	1/1	1/1
Lower clamp bound	1/1	1/1	1/1	1/1	1/1
Issues resolved	6/6	6/6	6/6	6/6	5/6
Composite	0.9823	0.9917	0.9928	0.9928	0.8333

No auxiliary held-out or regression tests were executed in any outcome. Empty auxiliary groups received passing defaults and contribute no independent robustness evidence. Reference-relative patch size and locality are not correctness certificates. Replaying one patch is not another independent generation, and no repeated-run variability is established.

B.3 To-do application and final feature positions

The brief requests one full-stack to-do application. Table 6 lists its five checks: a rendered task-list page, an input form, listing tasks, creating a task, and reading that task back in the same session. The last check does not establish persistence across process restarts. All candidates recorded successful builds. There are five applications and 25 checks, not 25 independent tasks.

Table 6: To-do application: one means the check passed and zero means it failed. All five candidates recorded successful builds. Read after write means a newly created task is returned in the same session, not persistence across process restarts. These are five candidate applications with five checks each, not 25 independent tasks.

Final feature check	Codex raw	Codex + Cortex	Claude raw	Claude + Cortex	OpenCode reference
Page title	1	1	1	1	1
Input form	1	1	1	1	1
List tasks	1	1	1	1	1
Create task	0	1	1	1	1
Read after write	0	1	1	1	1
Passed features	3/5	5/5	5/5	5/5	5/5

Figure 7 displays the same final binary outcomes in their checklist order, with normalized position from first to last feature. It does not depict a completion-decay curve: there are no successive builds, evolving requirements, or timestamped milestones in this summary. A collapsed interval from a single candidate would not establish repeatability.

The surrounding historical summary describes a chat app and a larger feature inventory, whereas the retained task and individual outcomes consistently identify these five to-do checks. Separate chat illustrations and the current chat protocol in Section 6.3 are not part of this scored evidence.

B.4 Storefront components and composite

Two pools concern the same fashion-shopping storefront brief. All candidates record successful build and serve checks and four screenshot entries. Existing artifacts were rescored rather than regenerated for this analysis. Table 7 contains the two-arm pool; Table 8 contains the separate five-arm pool.

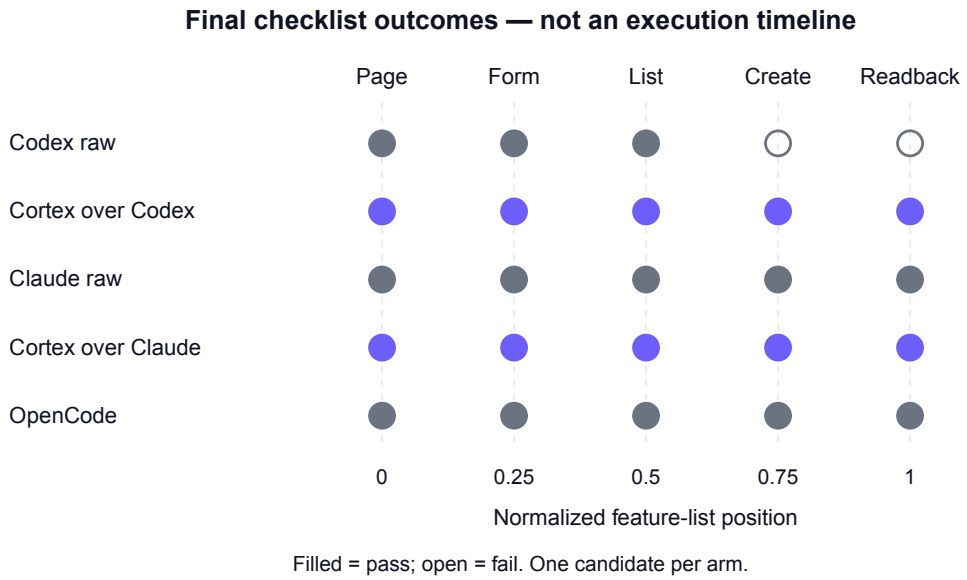

Figure 7: The five observed final feature outcomes in checklist order. Position is the feature index divided by four; it is not elapsed time, a repair pass, or a milestone observation. Table 6 defines the checks and provides the binary data. No interpolation or completion-decay curve is supported by these observations.

The eight composite weights, in table order before the composite row, are 0.26 for functional behavior, 0.14 for VERTEX, 0.09 for progressive-web-app and accessibility checks, 0.16 for visual similarity, 0.12 for code and architecture, 0.08 for robustness, 0.06 for code health, and 0.09 for static security. They sum to one and apply behind the build gate in Section 5.2. The separately reported UX score is not another term.

Functional checks probe requested behavior; VERTEX compares capability and architecture descriptors; PWA/accessibility checks are coarse structural proxies; visual similarity compares screenshots. Code/architecture, robustness, code-health, and static-security scores are bounded evaluator summaries, not exhaustive correctness, resilience, or vulnerability certifications. Section 6.3 details these limitations. The recorded visual backend is CLIP ViT-B/32 and the descriptor embedding backend is all-MiniLM-L6-v2. These are recorded identities, not an independent attestation of the complete historical environment.

Table 7: Two-arm storefront pool. One built candidate per arm; these are retained, subsequently rescored outcomes. All scores are in [0,1]. Component meanings and weights are specified in Appendix B.4; proxy scores of one do not certify exhaustive correctness. The composite uses authored-reference VERTEX, not the separate estimated-reference diagnostic.

Component	Codex raw	Claude raw
Functional	0.3636	0.7273
VERTEX	0.4309	0.5271
PWA / accessibility	0.8750	0.6250
Visual similarity	0.5930	0.6763
Code / architecture	0.7682	0.7628
Robustness	1.0000	1.0000
Code health	1.0000	1.0000
Static security	1.0000	1.0000
Composite	0.6507	0.7489

Table 8: Five-arm storefront pool. One built candidate per arm; these are retained, subsequently rescored outcomes. All scores are in [0,1]. Component meanings and weights are specified in Appendix B.4; proxy scores of one do not certify exhaustive correctness. The composite uses authored-reference VERTEX, not the separate estimated-reference diagnostic.

Component	Codex raw	Codex + Cortex	Claude raw	Claude + Cortex	OpenCode reference
Functional	0.5455	0.4545	0.4545	0.4545	0.7273
VERTEX	0.5214	0.4776	0.4512	0.4199	0.5070
PWA / accessibility	1.0000	0.8750	0.7500	0.6250	0.6250
Visual similarity	0.6154	0.6138	0.6509	0.6242	0.6605
Code / architecture	0.7328	0.7328	0.7563	0.7519	0.6591
Robustness	1.0000	1.0000	1.0000	1.0000	1.0000
Code health	1.0000	1.0000	1.0000	1.0000	0.6000
Static security	1.0000	1.0000	1.0000	1.0000	1.0000
Composite	0.7212	0.6799	0.6737	0.6533	0.7071

Figure 8 gives a common-scale view of all these components. The composite uses authored-reference VERTEX; the estimated-reference diagnostic in Appendix B.5 does not replace that component. The pools do not isolate governance: configuration, repair opportunities, and final-evaluation independence were not held fixed.

B.5 Authored and estimated reference ranks

The authored reference contains 13 capability descriptions and five architectural anchors. Estimated references use the public brief and the synthetic design vocabulary explained in Section 5.1.1. The pool diagnostic can also use descriptors supported by at least two other arms, excluding the candidate itself. This peer component is unavailable in the two-arm pool and can contribute in the five-arm pool, so reference construction differs.

Table 9 contains all score pairs underlying Figure 6. Both methods rank Claude above Codex in the two-arm pool; with two unequal observations, matching order alone yields correlation 1. The five-arm pool exchanges raw Codex and Cortex over Codex between ranks one and three, leaving the remaining ranks unchanged and yielding correlation 0.6.

Table 9: Exact VERTEX pairs and within-pool ranks (1 = highest) for Figure 6. Rank arrows mean authored to estimated rank, not improvement over time. The two pools share one storefront brief; seven pairs are not seven independent task samples. Peer-consensus descriptors are unavailable with only two arms.

Candidate pool	Configuration	Authored VERTEX	Estimated VERTEX	Rank change
Two-arm	Codex raw	0.4309	0.4142	$2 \rightarrow 2$
Two-arm	Claude raw	0.5271	0.5465	$1 \rightarrow 1$
Five-arm	Codex raw	0.5214	0.6026	$1 \rightarrow 3$
Five-arm	Cortex over Codex	0.4776	0.6465	$3 \rightarrow 1$
Five-arm	Claude raw	0.4512	0.5971	$4 \rightarrow 4$
Five-arm	Cortex over Claude	0.4199	0.5126	$5 \rightarrow 5$
Five-arm	OpenCode	0.5070	0.6159	$2 \rightarrow 2$

Both diagnostics concern one independently specified brief. No cross-brief holdout evaluation or correlation interval is reported. Similarity-rank agreement is not agreement on complete task correctness; peer-derived references can reproduce shared errors. These observations motivate further validation, not discarding authored references or claiming measured long-horizon performance.

Two-arm storefront pool

	Codex raw	Claude raw
Functional	0.36	0.73
VERTEX	0.43	0.53
PWA / accessibility	0.88	0.62
Visual similarity	0.59	0.68
Code / architecture	0.77	0.76
Robustness	1.00	1.00
Code health	1.00	1.00
Static security	1.00	1.00
Composite	**0.65**	**0.75**

Five-arm storefront pool

	Codex raw	Codex + Cortex	Claude raw	Claude + Cortex	OpenCode reference
Functional	0.55	0.45	0.45	0.45	0.73
VERTEX	0.52	0.48	0.45	0.42	0.51
PWA / accessibility	1.00	0.88	0.75	0.62	0.62
Visual similarity	0.62	0.61	0.65	0.62	0.66
Code / architecture	0.73	0.73	0.76	0.75	0.66
Robustness	1.00	1.00	1.00	1.00	1.00
Code health	1.00	1.00	1.00	1.00	0.60
Static security	1.00	1.00	1.00	1.00	1.00
Composite	**0.72**	**0.68**	**0.67**	**0.65**	**0.71**

Recorded scores: pale 0 → dark 1. Equal shading does not mean equal validity.

Figure 8: Recorded storefront score profiles, rounded to two decimals for display. Tables 7–8 retain four-decimal values and distinguish the two separately captured candidate pools. Component scores are proxies, not success probabilities; perfect recorded robustness or static-security values do not establish exhaustive testing or absence of vulnerabilities. The composite is a weighted, build-gated combination, not an additional independent measurement.