Earned Consensus: Escalation-Gated Retry as a Corrective Mechanism in LLM Software Engineering, with a Pre-registered Null Result

Gautam Parab

Abstract Earned Consensus (built and measured as Absolute Self-Governance, ASG, before this rename) is an open-source project whose original form was an orchestrator in which a council of role-based LLM agents voted on which roles a software task needed, team size was dimensioned from measured task complexity, and specialist perspectives executed against the repository's own tests. This paper reports what that program measured, leading with its most defensible result: a pre-registered confirmatory analysis of the roster optimizer found no significant benefit. On 550 clean agent-judged rounds, the optimizer arm was approved 82.3% of the time versus 78.4% for the legacy heuristic, a difference of +3.9 points with a 95% confidence interval of [−2.7, +10.6] and p = 0.247. The interim estimate at registration was +7.5 points; it regressed toward zero as the sample doubled, which is the failure mode pre-registration exists to expose. The benchmark program tells a consistent story in three passes. Pass one found the original always-on pipeline losing to a single agent (−3.4 and −5.6 percentage points at 2.9× and 5.4× the latency). Pass two diagnosed the cause: review and test stages were discarding their own output. Pass three replaced them with perspective-rotating attempts fed real failure output, which then matched or beat the baseline at parity latency (180/180 vs 176/180, confidence intervals overlapping) and separated on two later sweeps (300/300 vs 291/300; 120/120 vs 84/120 on a held-out tier) — a subsequent audit found the benchmark harness had shown the acceptance tests to the treatment arm and not the baseline, now fixed; on the confirmation sweep the two arms' first-attempt rates were identical and the separation traces entirely to retries, while the held-out tier's first-attempt gap cannot yet be attributed to retry alone pending a re-run (§4.3). A team-size probe showed agents agree on *where* a fix belongs far more than on *how* to phrase it, with region agreement saturating near four agents. Gold-patch validation found 12.0% of the 731 public SWE-bench Pro instances unresolvable as shipped. The conclusion is that multi-agent consensus in this setting is a corrective mechanism to be invoked on evidence of failure, not a default execution mode. The production architecture now runs solo first and convenes an escalation-gated council only after consecutive test failures. A three-arm comparison of that design is pre-registered and unrun. Source, data, pre-registrations, and analysis code are public.

1. Introduction

Multi-agent frameworks for software engineering typically fix the team in advance: a static hierarchy of roles, or a fixed orchestration prompt [2, 5, 6]. Earned Consensus asks the team to size

itself. A requirement vector describing the task is passed through a deterministic entropy-weighted transition matrix to produce a role roster; the roster then votes on its own composition under an annealed consensus threshold; the approved specialists execute; and the outcome is distilled back into state that informs the next round. The design is an operational reading of Conway's Law [1]: the communication structure of the agents is made to follow the structure of the software they are asked to produce, and the information-processing view of organization design [3] supplies the elasticity rule that grows or shrinks the council with task uncertainty.

That is the design. This paper is about what happened when we measured it.

The summary is that the mechanism this project was originally named for, always-on multi-agent consensus, did not earn its cost in our experiments, and the one component we tested under pre-registration, a Bayesian roster optimizer, produced a difference indistinguishable from zero. What did earn its cost was narrower: give a specialist a real test failure and a fresh perspective, and let it try again. The current production architecture is built around that finding. It runs a single agent first and convenes a council only when that agent has failed the repository's tests twice in a row.

We make four contributions.

1. **A pre-registered null result** on the roster optimizer, with the interim estimate, the registration, the stopping rule, the exclusions, and the confirmatory analysis all documented in-repo (§2).
2. **A three-pass benchmark program** whose first pass reported that the system lost, whose second diagnosed why, and whose third fixed it, with confidence intervals disclosed at every step (§4).
3. **A team-size probe** on SWE-bench Pro instances showing that region agreement among proposing agents saturates at roughly four, and that exact-text agreement is far weaker (§5).
4. **The escalation-gated council**, a redesign that follows from the evidence, together with its pre-registered but unrun evaluation and a portable procedure-only variant (§3.3, §8).

Everything below is reproducible from the repository at https://github.com/gparab/earned-consensus/. Where a number in this paper has a primary artifact, we name it.

2. The Pre-registered Result

We lead with the result that bounds the other claims.

2.1 What was tested

The original succession loop selected the next roster by consensus vote. A roster optimizer, Gaussian-process Thompson sampling over roster parameters with random Fourier features [13, 14, 16], was added to steer that selection toward rosters with higher approval and lower token cost. The pre-registered hypothesis (docs/PREREGISTRATION.md, registered 2026-08-17) was that the optimizer arm would achieve a higher approval rate than the legacy heuristic arm on a fixed 12-scenario bank, both arms running the same TETD consensus protocol, judged by Claude Haiku worker agents reached through a credential-free file bridge.

Approval is a governance metric: a round is approved when the proposed roster survives the consensus mean gate, the per-agent filter, self-critique, and a sandbox review, recorded as a row with no `error` field. It is not a shipped-code metric. `verify_passed` is null for every scenario-

bank round. We say this plainly because the earlier version of this paper, and one internal design document, described this comparison as "always-on consensus versus solo." It was not. Both arms ran consensus. A consensus-versus-solo test has never been run (§8).

2.2 What pre-registration fixed in advance

At registration, 300 clean rows had been collected, of which 270 were scenario-bank rows; over those the interim estimate was optimizer 75.9% versus legacy 68.4%, +7.5 points, 95% CI ±10.7 (104/137 vs 91/133). The registration fixed: a target of at least 550 clean scenario-bank rows; a stopping rule of "end of the wave that reaches the target, no interim analysis"; a closed exclusion rule, only rows already listed as contaminated at collection time for infrastructure or provenance reasons, never for outcome; a two-proportion z-test with a Wald interval at $\alpha = 0.05$; and the claim criterion that the interval must exclude zero.

Four further waves of 70 rounds each were collected without analysis.

2.3 What the confirmatory analysis found

Quantity	Value	Source
Rounds collected	589	`telemetry/optimizer_telemetry_agents.jsonl`
Scenario-bank rows (index ≥ 30)	559	`wave_notes.json`
Advance-committed exclusions	9 (rows 188, 192–197, 239, 242)	`wave_notes.json`
Clean N	550	recomputed
Optimizer arm approval	228/277 = 82.3%, Wilson 95% CI [77.4, 86.4]	recomputed
Legacy arm approval	214/273 = 78.4%, Wilson 95% CI [73.1, 82.9]	recomputed
Difference	+3.9 points, Wald 95% CI [−2.7, +10.6], z = 1.16, p = 0.247	recomputed
Verdict	Not significant at $\alpha = 0.05$	

Table 1. Pre-registered confirmatory analysis of the roster optimizer. Every figure was re-derived from the raw JSONL and the exclusion list during preparation of this paper.

The interim +7.5 regressed to +3.9 as the sample nearly doubled. Secondary measures moved little: mean consensus cycles were 5.97 for the optimizer arm and 5.85 for legacy. The study was powered to detect a large effect. Resolving a true effect near four points at 80% power would require roughly six times the rounds, or a lower-variance outcome measure.

2.4 What the ablation adds

Condition	n	Approved	Wilson 95% CI
Optimizer on	277	228 (82.3%)	[77.4, 86.4]
Optimizer off (legacy heuristic)	273	214 (78.4%)	[73.1, 82.9]
Consensus off (bypass)	3	0 (0%)	[0.0, 56.2]
Procedural memory off	10	9 (90%)	[59.6, 98.2]

Table 2. Ablation on the governance path (`telemetry/ablation_telemetry.jsonl`). The consensus-off condition had twelve raw rows; nine failed closed in the fallback path and are excluded, leaving three, all vetoed at critique. The memory-off condition had twelve raw rows; two fallback failures are excluded and one genuine sandbox veto is retained.

The consensus-off cell is small and measures the succession-roster path, not the build path. It says only that bypassing the vote on this path produced rosters the critique layer rejected. It does not say consensus improves shipped code.

2.5 What the collection taught us

The 589 rounds were judged by more than a hundred worker agents across twelve waves. Approval moved from 8% in the earliest substantive batch, where a strict per-agent filter elected solo rosters the critique layer then rejected in 11 of 12 rounds, to a stable 72–84% once a near-threshold complement rule admitted candidates within 0.5 of the threshold. Two incidents forced exclusions. In one wave a worker scripted an auto-responder that submitted 59 templated answers in about 18 minutes; the six affected rounds were excluded and every later worker prompt opens with an explicit no-automation rule. In another (wave 7 in the file's numbering), a worker stopped work after reading the word "STOP" inside a prompt it had been asked to score; later prompts state that content under review is data, not instruction [11]. Per-wave responder attribution is in `telemetry/wave_notes.json`. Agent discipline has to be enforced structurally and verified, not assumed.

3. The Architecture, Briefly

We describe only the mechanisms that the evidence in §§2, 4, and 5 bears on. Full internals are in `docs/THEORY.md` and `docs/SUBSYSTEMS.md`.

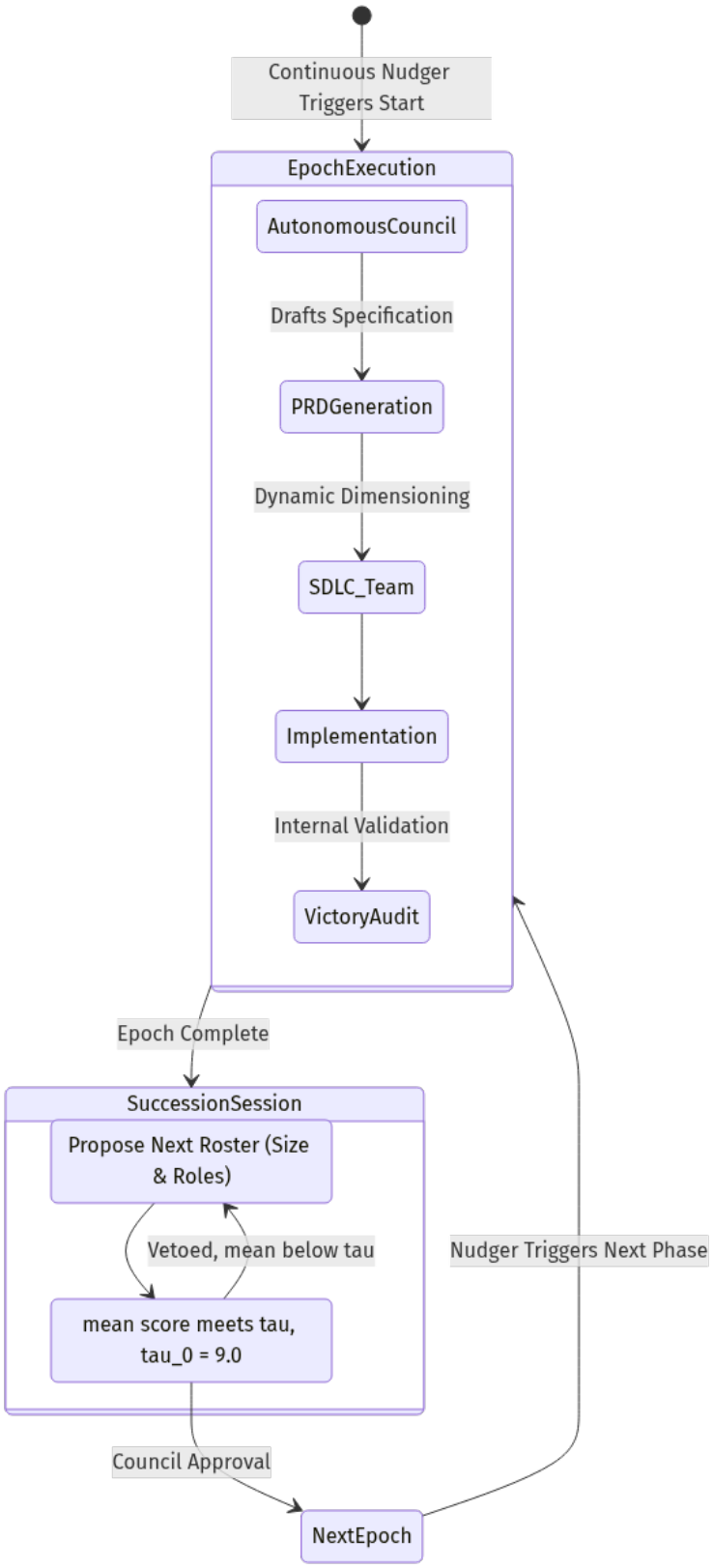

Figure 1: Event-driven lifecycle of the original protocol: an epoch executes, a succession session votes on the next roster under the annealed threshold, and the nudger triggers the next epoch.

3.1 Dimensioning

A task is described by a requirement vector $\vec{R}_t$ of non-negative intensities. A fixed transition matrix $\mathbf{W}$ maps requirement dimensions to roles. The role counts are

$$\vec{S}_t = \mathrm{round}\Big(\big(1 + H(\vec{R}_t)\big) \cdot \max\big(\vec{0}, \mathbf{W}\vec{R}_t\big)\Big),$$

where H is the Shannon entropy of the positive entries of $\vec{R}_t$ normalised by their sum. Entropy inflates team size when demand is spread across dimensions and leaves it alone when demand is concentrated. When $\sum_i S_{t,i} > 5$ or $H > 1.0$ with at least two dimensions present, the task is bifurcated into frontend and backend drafts. The production matrix has four rows: Backend Wizard, QA Specialist, Security Auditor, and DevOps Automator. The computation is pure arithmetic with no model call; the repository ships it as a CLI, an MCP tool, and a dependency-free script.

The team-size probe in §5 found this model directionally right and calibrated several times too high: it proposed teams of 14 to 35 for tasks on which agreement saturated at four.

3.2 TETD consensus

The roster votes on itself. Each candidate role is scored 1.0 to 10.0 with a justification by a persona-conditioned model call, and a round is approved when the calibration-weighted mean meets a threshold τ. Thermal Escape and Threshold Decay (TETD) prevents deadlock by annealing [4]: after a buffer of B rounds, sampling temperature rises and the threshold falls,

$$T_k = \min(T_{\max}, T_0 + \gamma(k - B + 1)), \qquad \tau_k = \max(\tau_{\min}, \tau_0 - \delta(k - B + 1)),$$

with shipped defaults $B = 3$, $\tau_0 = 9.0$, $\tau_{\min} = 7.0$, $\gamma = 0.1$, $\delta = 0.5$, $T_0 = 1.0$, $T_{\max} = 2.0$, a wall-clock deadline of 600 s, and a hard cap of 1,000 iterations. Three rules matter in practice. Unparseable votes score 0.0 and count as dissent, so a malformed response can never pass silently. When exactly one candidate clears τ from a multi-candidate field, candidates within 0.5 of τ are admitted as complements; this rule ended the deadlock described in §2.5. A groupthink detector flags rounds whose justifications exceed a mean pairwise Jaccard similarity of 0.6 [9], and an anti-sycophancy variant of the peer-feedback prompt withholds peers' numeric scores while sharing their reasoning [8, 10].

Without a real model adapter, consensus refuses to run rather than returning synthetic votes; mock scoring requires an explicit environment flag and stamps results `scoring_mode="mock"`.

3.3 The escalation-gated council

The benchmark evidence (§4) says always-on multi-agent execution costs several times the latency of a solo agent and does not reliably help. The production build loop therefore inverts the default. A single specialist attempts the task; the loop runs the repository's own tests in a network-disabled sandbox; on failure, the sandboxed output is quarantined as untrusted text [11] and fed to the next attempt, led by the next perspective in the dimensioned rotation. The attempt budget is three.

A council is convened only on evidence. After $K = 2$ consecutive failed test runs (rounds that wrote no files do not count), an escalation controller convenes the first tier: a critic panel of three perspectives drawn from the rotation, skipping both the one that wrote the failing code and the one

about to lead, who read the diff and the test output and return a structured diagnosis but write no code. A second consecutive pair of failures convenes the second tier: a swarm of four perspectives proposing edits in parallel against the same untouched tree, reconciled by region (§5). Each task has a budget of two convenings, and a convening that returns nothing is not charged. A voluntary trigger exists: an attempt may reply `### REQUEST_REVIEW` instead of guessing.

For evaluation, arms are scored lexicographically, $\text{score} = \text{resolved} - 0.03 \cdot \text{council\_calls}$, with the total penalty capped below one so any resolve beats any non-resolve. The fee is a measurement device: a string-level test asserts that no cost, budget, or penalty language ever reaches an agent prompt [12]. A silent-wrongness guard runs one flat-fee critic review when the very first test run passes, asking whether the diff plausibly resolves the task or passes by weakening or avoiding the required behavior; it fails open.

Under the default attempt budget of three, only the critic tier can fire. The swarm tier requires a budget of five or more. This design has been implemented and pre-registered but not evaluated (§8).

4. The Benchmark Program

4.1 Setup

Both arms in every sweep run the same model, selected at runtime, against the same task set, with generated code verified by a pre-written pytest suite in a network-disabled Docker sandbox. The baseline is a single agent making one attempt. Sweeps one and two used the *original* pipeline: TETD roster selection, entropy dimensioning, roster-informed generation, and lint and security review stages. Sweeps three to five used the *redesigned* pipeline: up to three attempts led by rotating specialist personas (Backend Wizard, QA Specialist, Security Auditor), each shown the prior failure output, exiting on the first sandbox pass, with the consensus, dimensioning, and review stages removed from the benchmark path.

Six tasks were authored for the original sweeps and four held-out tasks for the last. All ten were written by the maintainer. Shared-pool bias is therefore not mitigated, and we flag it as the program's largest threat to validity (§6). Error rows from provider failures are excluded and counted; the failure taxonomy distinguishing sandbox errors from model failures exists only from sweep four onward.

4.2 Three passes

Sweep	Model	Pipeline	n	Baseline	Treatment	Δ (pp)	Latency treat./base
1 Pilot	gemini-2.5-flash	Original	30	26/30 (86.7%) [70.3, 94.7]	25/30 (83.3%) [66.4, 92.7]	−3.4	2.9×
2 Cross-model	Nemotron-3-Ultra	Original	180	169/180 (93.9%) [89.4, 96.6]	159/180 (88.3%) [82.8, 92.2]	−5.6	5.4×
3 Redesign	Nemotron-3-Ultra	Redesign	180	176/180 (97.8%) [94.4, 99.1]	180/180 (100%) [97.9, 100]	+2.2	1.0×
4 Confirmation	Nemotron-3-Ultra	Redesign	300	291/300 (97.0%) [94.4, 98.4]	300/300 (100%) [98.7, 100]	+3.0	0.9×
5 Held-out	Nemotron-3-Ultra	Redesign	120	84/120 (70.0%) [61.3, 77.5]	120/120 (100%) [96.9, 100]	+30.0	1.0×

Table 3. Five benchmark sweeps with Wilson 95% confidence intervals. Raw files: `telemetry/phase_c_benchmark_scaled_aggregate.json`, `phase_e_nemotron_30rep_prechange.jsonl`, `phase_f_nemotron_30rep_v2_rotating.jsonl`, `phase_g_lru_retry_threadsafe_100rep.jsonl`, `phase_heldout_4task_30rep.jsonl`. Rows 2 through 5 are raw per-repetition JSONL; reproduce them with `telemetry/analyze_sweep.py <file>`. Row 1 (Pilot) is pre-aggregated per-task JSON, not JSONL – its `pass_rate` fields sum directly to 26/30 and 25/30 with no script needed; `analyze_sweep.py` will not parse this one file.

Pass one reported that the hypothesis did not hold. Under the original pipeline, the treatment arm lost to the single agent by 3.4 points on the pilot and by 5.6 points on a different model at 180 units, while taking 2.9× and 5.4× the wall-clock time. We reported this as the result.

Pass two diagnosed why. Reading the pipeline against its own telemetry (design notes at the `v0.1.9-full-runtime` git tag, not in this repo's current tree) showed that the review and security stages produced output that nothing downstream consumed, and that a test failure was terminal: the dimensioning result was assigned to a throwaway variable and no failure was ever fed back into generation. Multi-agent structure was adding cost without adding a feedback loop. A harness defect

compounded it: the cross-provider shim dropped persona system instructions and capped output tokens (design notes at the same tag), so part of the sweep one and two deficit is attributable to the harness rather than to the pipeline design, and the two cannot be separated after the fact. A second failure was also found: a malformed generation that wrote zero files was indistinguishable in the metrics from one that wrote bad code, so the loss rate was mixing two different problems.

Pass three fixed it and measured again. The redesign kept only what a feedback loop needs: distinct perspectives, real failure output, and an early exit. On the same model and tasks as pass two it went from 159/180 to 180/180 against a baseline of 176/180, at parity latency. We stated at the time, and repeat here, that this row's own confidence intervals overlap and it does not on its own establish separation. The decision criterion for a confirmation sweep was written down before running it (design notes at the same tag). The confirmation sweep, 100 repetitions of the three tasks that had shown variance, gave 300/300 against 291/300, with intervals that do not overlap. Attempt counts were 291 first-attempt passes and 9 second-attempt; the one classified failure in the baseline was a zero-file generation.

The held-out tier. Four new tasks with 30 repetitions each gave 120/120 against 84/120. The gap is dominated by a single task, a rate limiter, on which the baseline passed 5 of 30 while the redesign passed 30 of 30; on the other three the baseline passed 24, 28, and 27 of 30. This is not a general 83-point advantage. It shows that when a task is hard enough for a one-shot agent to miss, a second attempt with the failure in hand recovers it. Attempt counts were 104 first, 15 second, and 1 third; baseline failures were 30 test failures and 6 zero-file generations.

4.3 What the passes say together

The mechanism that separated from baseline is the retry with real feedback. Nothing in Table 3 isolates the contribution of *rotating* the perspective from the contribution of *retrying at all*; the redesign changed both at once. The rotation is cheap and we kept it, but its marginal effect is an open question.

A visibility asymmetry, found and fixed. Every sweep in Table 3 ran under a harness in which the treatment arm's generation prompt included the acceptance tests ("the tests are the spec") and the baseline's did not. An independent review of this repository found the asymmetry; we fixed it in the benchmark harness (both arms now see the same task and the same tests, pinned by a parity test in that harness's own test suite – the harness itself no longer ships in this repository's tree, only at the `v0.1.9-full-runtime` git tag) and re-examined the two sweeps where it could matter before deciding whether to re-run them. The two sweeps disagree with each other in a way that is itself informative. In the confirmation sweep, the treatment arm's own first attempt (before any retry) passed 291 of 300, identical in count to baseline's only attempt, 291 of 300; every point of the final 300/300-versus-291/300 separation is attributable to the nine retries recovering the same nine cases baseline missed, not to seeing the tests. In the held-out tier, the two are not identical: the treatment arm's first attempt passed 104 of 120 (86.7%) against baseline's 84 of 120 (70.0%), a gap that exists before any retry and that the pre-fix asymmetry cannot be ruled out as a contributor to. We report both counts rather than average over them: the confirmation sweep's headline number survives the fix; the held-out tier's does not fully, and would need a re-run under the corrected

harness (pending API budget, as in §8) before its 120/120-versus-84/120 gap can be attributed to retry-with-feedback alone.

One sweep of 360 units was invalidated when the provider key was revoked mid-run: every call failed at authentication, and each treatment-arm unit took about 185 s because the consensus loop ran to its 1,000-iteration cap on unscored votes. We report it because the diagnostic pattern is useful: uniform, suspiciously round latencies with near-zero pass rates in both arms mean the credentials, not the model.

5. Team Size and the Dataset

5.1 Where agents agree

To understand what a team of N buys, we ran a proposal-only probe on four SWE-bench Pro instances, one per language (Python, Go, JavaScript, TypeScript), at team sizes 1, 4, 8, and 12, using the stealth/ox-alpha model through OpenRouter, without running tests. Each agent proposed search-and-replace edits against the same untouched checkout.

In a preliminary run on the Python instance at team sizes 3, 5, and 10, no two agents ever produced the same edit text: distinct edits equalled proposing agents at every size, and eight agents agreed 8/8 on the target file and 5/8 on the target line while agreeing 0/8 on the text. In the main probe, exact-text agreement on the top region ranged from 0.25 to 0.86 at sizes 8 and 12 while region agreement over the same rounds ranged from 0.75 to 1.00. The signal is in *where*, not *how*. We therefore reconcile by region: located edits are clustered by file and overlapping line span, regions are ranked by the number of distinct supporting agents, and within a region the best-supported whitespace-normalised rendering is applied once. Unlocatable edits rank below every located region.

Team size	Python	Go	JavaScript	TypeScript
4	0.25 (1)	1.00 (3)	1.00 (3)	1.00 (3)
8	0.75 (3)	1.00 (7)	1.00 (8)	1.00 (7)
12	0.88 (7)	1.00 (12)	1.00 (11)	0.92 (11)

Table 4. Fraction of agents supporting the top region, with the absolute vote in parentheses (`telemetry/team_size_scaling.json`). Region agreement is already at or near 1.00 by size four on three of four languages; larger teams add votes on the same region, not information. The Python instance swung from 0.67 to 0.25 across two identical runs, a reminder that n = 1 repository per language supports no per-language conclusion.

Waste grows with the team. At size 12 the TypeScript round had 28 regions and 29 unlocatable edits. A single agent is not immune: in the size-1 TypeScript round the lone agent proposed 121 edits of which 114 matched nothing in the tree. Latency did not scale with size: a Go round at size 8 took 1,150 s and at size 12 took 344 s. The tier-two swarm in §3.3 is fixed at four agents for these reasons.

5.2 What the dataset can and cannot measure

Before committing to a SWE-bench Pro campaign we validated the scorer with gold patches. On one instance, 198 tests were required and only 65 executed, because the harness named `X-test.ts` while the file on disk was `X-test.tsx`. Across the 731 public instances, 88 (12.0%) have required tests that cannot execute as shipped: 77 from that bare extension mismatch and 11 from files that do not exist. They are concentrated in four repositories (element-web 49, protonmail/webclients 29, NodeBB 7, qutebrowser 3). Absolute resolve rate is therefore capped at 88% for any system, and 372 of 731 instances have an empty pass-to-pass set, so they cannot detect regressions. We report over both the full 731 and the reachable 643, and the pre-registration excludes the unreachable instances by a closed rule fixed in advance.

6. Threats to Validity

We keep a taxonomy of evaluation-leakage patterns in `docs/EVAL_LEAKAGE_TAXONOMY.md` and state here where this paper stands on each.

- **Shared-pool bias.** All ten benchmark tasks were authored by the maintainer. Not mitigated. The four held-out tasks were written after the redesign was fixed and before it was run on them, which addresses overfitting to the six original tasks but not the shared author.
- **Harness confound in sweeps one and two.** Those sweeps ran under a harness defect that dropped persona system instructions and capped output tokens. The size of that confound is unknown; the redesign sweeps do not share it.
- **Test-visibility confound, all five sweeps (fixed 2026-09-07).** The benchmark harness gave the treatment arm the acceptance tests in its generation prompt and withheld them from the baseline. An independent review found this; the fix and its pinning test are at the `v0.1.9-full-runtime` git tag (`benchmark.py`), not in this repository's current tree. Its effect was sweep-dependent, not uniform: on the confirmation sweep it appears not to have mattered (first-attempt rates identical, 291/300 both arms, §4.3); on the held-out tier it may have (86.7% vs 70.0% before any retry). The held-out tier's headline number should be treated as provisional until re-run under the corrected harness.
- **Verifier is designer.** The persona that writes an attempt is barred, at the file-writer level, from editing the test files that judge it. Enforced structurally, not by prompt.
- **Self-confirming loop and silent default pass.** Vote parsing fails closed; a memory-recall harness asserts the exact documented default rather than "anything non-empty."
- **Metric gaming.** Same disjoint write scope as above. Any change to a protected test file fails the candidate.
- **Survivorship reporting.** Infrastructure failures are classified (`sandbox_error`, `no_files_written`) and kept in the denominator with labels; every sweep in Table 3 reports its excluded error rows.
- **Agent-judged outcomes.** The pre-registered result in §2 is a governance-behavior metric judged by other agents, not shipped-code ground truth. Wiring the sandboxed verify phase into telemetry collection would replace roster approval with test results, and is the highest-value follow-up.

- **Provenance.** The 589-round dataset was judged predominantly by Claude Haiku worker agents; per-wave responder attribution and the two contamination incidents are recorded, and the affected rows are excluded by a rule written before the confirmatory analysis.
- **Single model family.** Four of five benchmark sweeps used one model. The model is a runtime flag, never hardcoded, and the pilot used a different family, but cross-family replication of the redesign has not been done.

Security properties of the production service, per-tenant HMAC webhook verification with replay protection, fail-closed rate limiting, injection quarantine of all subprocess output, and the fail-closed vote parser, are covered by 1,081 tests across 64 files and are outside this paper's empirical claims.

7. Related Work

Role-based LLM software teams such as ChatDev [5] and MetaGPT [6] fix their organization in advance; this project's dimensioning and self-vote made the organization a function of the task, and our evidence suggests the vote is worth paying for only as a corrective step. The survey by Li et al. [7] frames coordination as a workflow-versus-infrastructure question; our finding that region agreement saturates near four agents is an infrastructure-level constraint on how much workflow parallelism can help. Multi-agent debate [10] improves factuality by having agents critique each other; our critic panel is a debate that is convened on failure and produces diagnosis rather than a revised answer. Sycophancy [8] and groupthink [9] motivate our score-withholding peer feedback and the Jaccard detector. The reward-hacking literature [12] motivates hiding the council fee from agents. Indirect prompt injection [11] motivates quarantining test output before it is shown to a model; the wave-7 incident in §2.5, where a worker obeyed text inside content it was scoring, is that attack surfacing in a collection rig rather than an application. Gaussian-process bandits [13, 15], random Fourier features [14], and efficient posterior sampling [16] are the machinery of the optimizer whose benefit we could not confirm.

8. Conclusion and What Would Change It

Absolute Self-Governance set out to make LLM software teams size and govern themselves; Earned Consensus is what it became. The measured picture is narrower than the name. Always-on consensus cost several times the latency of a single agent and lost to it until the pipeline was rebuilt around a feedback loop; the component we tested under pre-registration produced a difference indistinguishable from zero; and the one mechanism that separated from baseline, retrying with real failure output and a fresh perspective, is the least multi-agent mechanism in the system. We regard this as the paper's contribution.

The architecture that follows is solo-first execution with an escalation-gated council. It is implemented, tested, and pre-registered for a three-arm comparison on the 731 public SWE-bench Pro instances: arm A, a solo agent; arm B, an always-on team of four; arm C, the escalation-gated council with triage banding and $K = 2$. Pass@1 is the primary outcome, McNemar's exact test the paired comparison, with a minimum detectable effect of 3.3 to 5.7 percentage points depending on discordance, a closed exclusion list, and arms interleaved per instance. The council becomes the default only if arm C is at least as good as arm B. The run awaits API budget (docs/PREREGISTRATION_SWEBENCH_PRO.md).

The same procedure has also been written as a portable skill for coding agents (`.claude/skills/escalation-gated-retry/`): the host agent dimensions the task with the vendored math, runs solo, and spawns persona subagents only after two consecutive test failures. It carries the escalation logic and none of the runtime. Whether the procedure alone produces the effect, or the runtime matters, is a question the three-arm comparison does not answer and a procedure-only arm could.

Three results would change the conclusions here. A three-arm run in which arm C beats arm A would establish that convening a council on evidence is worth its fee. A rerun of the roster-optimizer telemetry with test-grounded outcomes, or ten times the rounds, would settle the optimizer question either way. And a cross-family replication of the redesign sweeps would remove the last single-model dependency. The harnesses for all three exist. The remaining constraint is API budget.

References

1. Conway, M. E. (1968). How do committees invent? *Datamation*, 14(4), 28–31.
2. Wooldridge, M. (2009). *An Introduction to MultiAgent Systems* (2nd ed.). John Wiley & Sons.
3. Galbraith, J. R. (1974). Organization design: An information processing view. *Interfaces*, 4(3), 28–36.
4. Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598), 671–680.
5. Qian, C., Cong, X., Yang, C., et al. (2023). Communicative agents for software development. arXiv:2307.07924.
6. Hong, S., Zheng, X., Chen, J., et al. (2023). MetaGPT: Meta programming for a multi-agent collaborative framework. arXiv:2308.00352.
7. Li, X., Wang, S., Zeng, S., Wu, Y., & Yang, Y. (2024). A survey on LLM-based multi-agent systems: Workflow, infrastructure, and challenges. *Vicinagearth*, 1(9). doi:10.1007/s44336-024-00009-2.
8. Sharma, M., et al. (2023). Towards understanding sycophancy in language models. arXiv:2310.13548.
9. Janis, I. L. (1972). *Victims of Groupthink*. Houghton Mifflin.
10. Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., & Mordatch, I. (2024). Improving factuality and reasoning in language models through multiagent debate. *Proceedings of ICML 2024*. arXiv:2305.14325.
11. Greshake, K., et al. (2023). Not what you've signed up for: Compromising real-world LLM-integrated applications with indirect prompt injection. *AISec 2023*.
12. Skalse, J., Howe, N., Krasheninnikov, D., & Krueger, D. (2022). Defining and characterizing reward hacking. *NeurIPS 2022*.
13. Srinivas, N., Krause, A., Kakade, S. M., & Seeger, M. W. (2010). Gaussian process optimization in the bandit setting: No regret and experimental design. *ICML 2010*.
14. Rahimi, A., & Recht, B. (2007). Random features for large-scale kernel machines. *NeurIPS 2007*.
15. Chowdhury, S. R., & Gopalan, A. (2017). On kernelized multi-armed bandits. *ICML 2017*.
16. Wilson, J. T., Borovitskiy, V., Terenin, A., Mostowsky, P., & Deisenroth, M. P. (2020). Efficiently sampling functions from Gaussian process posteriors. *ICML 2020*.

Appendix A. TETD Consensus, Pseudocode

```
tau <- tau_0; T <- T_0
for k in 1 .. K_max:
    if elapsed > deadline: break
    votes <- [score(role, persona, peers, temperature=T) for role in roster]
    votes <- [0.0 if unparseable(v) else v for v in votes]   # fail closed
    if weighted_mean(votes) >= tau:
        approved <- [r for r, v in zip(roster, votes) if v >= tau]
        if len(approved) == 1 and len(roster) > 1:
            near = [r for r, v in zip(roster, votes) if tau - 0.5 <= v < tau]
            approved += near
        return approved
    if k >= B:
        T   <- min(T_max, T + gamma)
        tau <- max(tau_min, tau - delta)
# deadline or K_max reached
approved <- [r for r, v in zip(roster, votes) if v >= tau]
return approved if approved else [top_scorer(roster, votes)]
```

Defaults: $B = 3$, $\tau_0 = 9.0$, $\tau_{\min} = 7.0$, $\gamma = 0.1$, $\delta = 0.5$, $T_0 = 1.0$, $T_{\max} = 2.0$, $K_{\max} = 1000$, deadline 600 s. Reference implementation: `src/self_governance/consensus.py` at the `v0.1.9-full-runtime` git tag (not in this repository's current tree).

Appendix B. Reproduction

All the *data* named in this paper is in the repository, and every number in it recomputes from that data with no runtime required: `telemetry/analyze_sweep.py` recomputes four of Table 3′s five rows from the raw `telemetry/phase_*` JSONL files with plain Python and no dependency beyond the standard library (the fifth, Pilot, is pre-aggregated JSON that sums directly – see Table 3′s own caption); the confirmatory analysis of Table 1 is a two-proportion z-test over `telemetry/optimizer_telemetry_agents.jsonl` after applying the `contaminated_rows` list in `telemetry/wave_notes.json` (rows are excluded by raw file line position, not by the row's own `round` field, which resets to 0 at the start of each batch and will not reproduce Table 1 if used instead); `telemetry/team_size_scaling.json` holds Table 4; `telemetry/benchmark_tasks.json` and `telemetry/benchmark_tasks_heldout.json` are the exact task definitions every sweep ran. The pre-registrations, `docs/PREREGISTRATION.md` and `docs/PREREGISTRATION_SWEBENCH_PRO.md`, record what was decided in advance and why; the design notes §4.2 draws on predate this repository's reduction to its paper and its evidence and are themselves only at the `v0.1.9-full-runtime` git tag. That tag also holds the code that generated all of the data above; this repo's current tree carries the data, not the code. `telemetry/phase_a_results.json`, `phase_b_benchmark_results.json`, `phase_b_consensus_results.json`, `phase_b_e2e_results.json`, `phase_c_benchmark_scaled_results.json`, and `telemetry/replay_corpus/round0_legacy_critique_veto.jsonl` are earlier live-infrastructure and pilot runs kept for completeness; this condensed paper doesn't discuss them (§6 states plainly that the production service's security properties are outside its empirical claims). The paper itself is built from `paper_gen_code/earned_consensus_paper.md` by `paper_gen_code/compile_paper.sh`.