LinearBench: A Framework for Measuring Convergence, Stability, and Feedback Dependence in Iterative Product Construction by Language Models

Momen Ghazouani
Chief Scientist, Setaleur Aplamda
momen.officiel@gmail.com

October 3, 2026

Abstract. Most evaluations of large language models score either a single attempt or the end state of one autonomous run. In practice, complex artifacts such as applications and games are built over several rounds of feedback, and the quality of that process is largely unmeasured: how quickly it converges, how much previously working behavior it destroys, and how much external guidance it needs. We introduce LinearBench, a framework for evaluating language models on multi-round build-and-refine tasks against hidden, weighted acceptance specifications. A deterministic user simulator supplies feedback at one of four levels of informativeness, the feedback ladder, which separates self-correction from feedback following. We define trajectory-level metrics: stable turns-to-acceptance under right censoring, a regression and churn decomposition, a headroom-weighted feedback efficiency, a parameter-free persistence-based Convergence Score, and a Self-Sufficiency Ratio that replaces naive autonomy ratios. We prove that summaries computed from the quality sequence alone cannot detect regressions, that the Convergence Score never exceeds the area under the quality curve, and that feedback efficiency is bounded above by one. A synthetic study with parameterized model profiles shows that conventional and proposed summaries can rank the same profiles differently, and that a naive autonomy ratio can label a weak profile as the most self-sufficient. We state falsifiable hypotheses, threats to validity, and a design for the planned empirical evaluation. This paper reports no evaluation of existing language models, and makes no claim about general intelligence.

Keywords: language model evaluation, multi-turn interaction, regression, user simulation, self-correction, benchmarks

1 Introduction

Language model benchmarks usually measure one of two things. The first is the quality of a single attempt, as in function-level code generation [1]. The second is the end state of one autonomous run, as in repository-level issue resolution [2] and tool-agent-user environments [3]. Both are useful, and both abstract away a pattern that dominates real use of capable models. A person asks for something complex, such as a game, a data tool, or an application, receives a reasonable first version, and then iterates with short reports of the form "the camera shakes when sprinting" or "the safe zone does not shrink correctly." Within a few rounds a strong system reaches a stable and complete product. A weaker one may stall, oscillate, or repair one behavior while silently breaking another.

Several lines of work are adjacent to this pattern. SWE-bench includes tests that must keep passing after a fix, which checks for regressions in the final state of a single run [2]. InterCode evaluates interactive coding with execution feedback [10], and Self-Debugging and CRITIC show that execution results or tool feedback can improve a model's own output [11, 12]. MINT measures how much models gain from tool use and from natural language feedback supplied by a simulated user [4]. Laban et al. show that models lose both aptitude and reliability in multi-turn settings where instructions are revealed gradually [5]. Work on self-correction reports that models often fail to improve their own reasoning without an external signal [6, 7]. What remains unmeasured, to the best of our knowledge, is the combination of three things: (i) the full trajectory over several rounds on an artifact with many interacting requirements, (ii) criterion-level regressions across rounds, meaning previously working behavior that stops working after a revision, and (iii) a controlled decomposition of how much of iterative success depends on the information the model receives rather than on the model itself.

LinearBench targets these three quantities. Its design rests on a hidden specification of weighted, checkable criteria, a fixed budget of K attempts (the first attempt followed by $K-1$ rounds of feedback), a deterministic user simulator, and a ladder of four feedback levels from no information to rich machine diagnostics.

Contributions.

1. A formal framework for iterative construction under a feedback policy, including visible and hidden criteria and gate semantics that avoid double counting (Section 3).
2. The feedback ladder, a controlled variable that

turns the vague notion of autonomy into a measurable gradient, together with the Self-Sufficiency Ratio (Section 3.3, 4).
3. A metric suite with proved properties: an insufficiency result for quality-only summaries, a parameter-free Convergence Score, and a bounded feedback efficiency (Sections 4 and 5).
4. An evaluation protocol with statistical design, simulator validation, and contamination controls (Section 6).
5. A synthetic study that exercises the metrics on parameterized model profiles and documents where simpler alternatives fail (Section 7).

Scope. LinearBench evaluates models as they are. It is not a training method and does not replace coding, reasoning, or knowledge benchmarks. We use "autonomy" only in an operational sense, as independence from external information within the protocol. Whether this relates to broader notions of intelligence is a question we leave to discussion and do not build into the definitions. The present paper is a measurement proposal: all numerical results come from a synthetic simulation and none from language models.

2 Related Work

Single-attempt and end-state evaluation. Function-level benchmarks score one generation per problem [1]. Repository-level and agentic benchmarks score the final state of an autonomous run against tests [2, 3]. SWE-bench also checks that previously passing tests still pass, which is a regression check on the end state of one run. None of these records how quality, or the set of passing criteria, evolves over successive rounds of feedback.

Multi-turn and interactive evaluation. MINT evaluates multi-turn interaction with tools and language feedback, uses GPT-4 to simulate the user, and reports that better single-turn performance does not guarantee better multi-turn performance [4]. Laban et al. compare single-turn and sharded multi-turn settings and separate aptitude from unreliability [5]. InterCode provides interactive coding environments with execution feedback [10]. LinearBench differs in its unit of analysis (a many-criteria artifact rather than a single answer), its deterministic non-LLM simulator, its explicit information ladder, and its criterion-level regression metrics computed across rounds.

Self-correction. Self-Refine and Reflexion show that iterative self-feedback can help on some tasks [8, 9], and Self-Debugging and CRITIC report gains from execution results and tool-based critiques [11, 12], while Huang et al. and the survey of Kamoi et al. argue that gains often depend on external signals or oracle information [6, 7]. The L0 versus L3 contrast in LinearBench places this question inside a single controlled protocol on complex artifacts.

Simulation, judging, and contamination. Simulated users and model judges improve reproducibility but introduce their own biases [13]. We therefore favor programmatic verification, restrict judge models to a different family from the model under test, and validate the simulator against humans. Contamination of popular tasks is a known threat to code evaluation [14], which motivates parameterized task instances and a private split.

Cost-aware and statistical practice. Agent evaluation that ignores cost can mislead [15]. We adopt Pareto reporting of quality against tokens, survival-style treatment of censored times [16], and cluster bootstrap intervals [17].

3 Framework

3.1 Tasks and hidden specifications

Definition 1 (Task specification)**.** A task is a tuple $(x, \mathcal{C}, w, G, V, \mathrm{pre})$. Here x is a natural-language request shown to the model, $\mathcal{C} = \{1, \dots, n\}$ is a set of criteria, and $w_i > 0$ with $\sum_i w_i = 1$ are weights. The set $G \subseteq \mathcal{C}$ contains the *gate* criteria (for example, launches without error), and $V \subseteq \mathcal{C}$ contains the *visible* criteria, which are stated in or directly implied by x. The remaining criteria $H = \mathcal{C} \setminus V$ are *hidden*. The map pre assigns to each non-gate criterion a set of prerequisite gates.

A verifier Ver maps an artifact a to a state $s \in \{1, 0, \perp\}^n$. For $i \in G$, $s_i \in \{0, 1\}$ records whether the criterion holds. For $i \notin G$, $s_i = \perp$ (not evaluable) if some gate in $\mathrm{pre}(i)$ fails, and otherwise $s_i \in \{0, 1\}$. We write $P(s) = \{i : s_i = 1\}$, $\mathcal{S}(s) = \{i : s_i = \perp\}$, and $q(s) = w(P(s)) = \sum_{i \in P(s)} w_i$. Given a threshold $\theta \in (0, 1]$, a state is *accepted* if $G \subseteq P(s)$ and $q(s) \geq \theta$. The $\perp$ value matters because a crash that disables thirty dependent criteria is one defect, not thirty, and Section 4 attributes it accordingly.

The visible/hidden split separates two abilities that a single hidden specification would conflate: inferring unstated requirements and repairing known defects. Results are reported on V, on H, and on all of $\mathcal{C}$.

3.2 Interaction protocol

For a model M, feedback level $\ell \in \{0, 1, 2, 3\}$, and budget K (that is, K attempts and $K - 1$ feedback messages), a run proceeds as follows (Figure 1). The model produces $a_1 \sim M(\cdot \mid x)$. For $t = 1, \dots, K$ the verifier computes $s_t = \mathrm{Ver}(a_t)$, and for $t < K$ the simulator emits $m_{t+1} = \pi_\ell(s_t, a_t)$ and the model produces a_{t+1} conditioned on the full history $(x, a_1, m_2, \dots, a_t, m_{t+1})$. Four design rules apply. First, all K turns are run even after acceptance, so

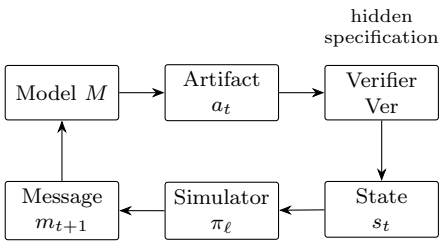


Figure 1: One round of the LINEARBENCH protocol. The verifier applies the hidden specification, and the simulator converts the resulting state into a message according to the feedback level ℓ. All K attempts are made regardless of acceptance.

that late regressions and persistence are observable. Second, the model has the same tools and sandbox at every level, so levels differ only in the external information supplied. Third, the first attempt a_1 is shared across levels for each (task, seed), which removes first-attempt variance from level comparisons. Fourth, the hidden specification is never revealed: the simulator describes symptoms, not criteria, weights, or scores.

3.3 The feedback ladder

Let $F_t = \{i : s_{t,i} = 0\}$ be the evaluable failing criteria at turn t, and let $\prec$ be the total order on $\mathcal{C}$ that places gates first, then larger weights, then smaller indices. The four levels are as follows.

- **L0 (self-review).** π_0 always sends a fixed generic instruction to review and improve the work.
- **L1 (vague).** π_1 always sends a fixed note that the result is not yet good enough.
- **L2 (specific).** π_2 sends one templated symptom description for the $\prec$-first element of F_t, or a neutral message if $F_t = \emptyset$.
- **L3 (automated).** π_3 sends machine-readable diagnostics (test output, logs, traces, screenshots) for every element of F_t.

Two observations guide interpretation. L0 and L1 are both state independent, so any difference between them reflects the pragmatic assertion that defects exist, not information about which ones. And L2 and L3 are not nested: a large diagnostic bundle can bury the salient symptom that a single sentence would surface. The ordering L0 $\le$ L1 $\le$ L2 $\le$ L3 in performance is therefore a hypothesis to be tested, not an assumption.

4 Metrics

Fix a run with states $s_1, \dots, s_K$, passing sets $P_t = P(s_t)$, and qualities $q_t = w(P_t)$.

4.1 Quality, persistence, and the Convergence Score

The area under the quality curve is AUC $= \frac{1}{K}\sum_{t=1}^{K} q_t$. It rewards early quality but cannot see whether the passing criteria are the same ones from turn to turn. We define the *persistent set*

$$Q_t = \bigcap_{u=t}^{K} P_u, \qquad q_t^p = w(Q_t),$$

the weight of criteria that pass at turn t and keep passing through the end of the horizon. The *Convergence Score* is

$$\mathrm{CS} = \frac{1}{K}\sum_{t=1}^{K} q_t^p. \tag{1}$$

A criterion that is fixed, broken, and fixed again contributes only from its final repair. The score is parameter free: regressions are penalized by construction and not through a weighted subtraction, which would double count them because AUC already falls when criteria are lost.

4.2 Turns-to-acceptance

Let $T = \min\{t \le K : \text{state } s_t \text{ is accepted}\}$, with $T = \infty$ (right-censored) if no such turn exists. Let $T^s = \min\{t : s_u \text{ is accepted for all } u \ge t\}$ be the *stable* time. We report the acceptance rate $\mathrm{AR@}K = \Pr(T \le K)$ and the restricted mean $\mathrm{RMTA}_K = \mathbb{E}[\min(T, K+1)] = \sum_{t=0}^{K} \Pr(T > t)$, which uses the survival function and handles censoring without discarding failed runs. The gap $\mathbb{E}[\min(T^s, K+1) - \min(T, K+1)]$ measures how fragile acceptance is.

4.3 Regression and churn

Define the gained weight $A_t = w(P_t \setminus P_{t-1})$. A passing criterion can leave P in two ways. It can fail while remaining evaluable (or be a gate that fails), which is a *root-cause regression* of weight $B_t = w\big((P_{t-1} \setminus P_t) \setminus \mathcal{S}(s_t)\big)$. Or it can become unevaluable because a gate failed, which is *shadowed loss* of weight $D_t = w\big(P_{t-1} \cap \mathcal{S}(s_t)\big)$. The weighted regression rate and churn are

$$\mathrm{RR}_t = \frac{B_t}{q_{t-1}} \ \ (0 \text{ if } q_{t-1} = 0), \qquad \chi_t = A_t + B_t + D_t, \tag{2}$$

and $\mathrm{RR} = \frac{1}{K-1}\sum_{t=2}^{K} \mathrm{RR}_t$. High churn with little net change indicates oscillation, whereas low churn with little net change indicates stagnation, and the two have very different practical meaning.

4.4 Feedback efficiency

The per-turn gap closure is $g_t = (q_t - q_{t-1})/(1 - q_{t-1})$ when $q_{t-1} < 1$. Averaging g_t directly is undefined

at the ceiling and heavy-tailed near it. We define instead

$$\mathrm{FE} = \frac{q_K - q_1}{\sum_{t=2}^{K}(1 - q_{t-1})}, \tag{3}$$

which weights each round by the headroom available at its start. A run is *eligible* for FE if $q_1 \le 1 - \eta$ (default $\eta = 0.1$), and across runs we pool numerators and denominators (a ratio of sums) instead of averaging ratios.

4.5 Feedback dependence

Because a_1 is shared, q_1 is the exact no-revision counterfactual for every level. Define the *revision gain* at level ℓ as $\mathrm{Gain}_\ell = \mathbb{E}[\mathrm{CS}_\ell - q_1]$ and the *feedback uplift* $U_\ell = \mathrm{Gain}_\ell - \mathrm{Gain}_0$. The *Self-Sufficiency Ratio* is

$$\mathrm{SSR}_\ell = \frac{\mathrm{Gain}_\ell}{\max_{\ell'} \mathrm{Gain}_{\ell'}}, \tag{4}$$

defined only when $\max_{\ell'} \mathrm{Gain}_{\ell'} > \delta$ (default $\delta = 0.02$); otherwise we report that no feedback-driven improvement was measurable. It can be negative, which indicates that self-revision lowers persistent quality. The naive alternative $\mathrm{CS}_0/\mathrm{CS}_3$ fails on weak models: a model that improves under no level of feedback has $\mathrm{CS}_0 \approx \mathrm{CS}_3 \approx q_1$, so the ratio is near one and suggests high independence when there is simply no improvement to be independent about. The denominator uses the best level instead of L3 because the ladder need not be monotone. Gains are averaged over runs before taking the ratio.

4.6 Reference trajectories

Three reference runs anchor the scale. The *null* run never changes the artifact, so $q_t = q_1$ and $\mathrm{CS} = q_1$. The *unit-rate oracle* repairs exactly the $\prec$-first failing criterion each turn and changes nothing else, giving a monotone trajectory q_t^*, a time T^*, and $\mathrm{CS}^* = \frac{1}{K}\sum_t q_t^*$. The *random-edit* control flips criteria independently with probabilities matched to the model's measured churn. We report the normalized score $\mathrm{NCS} = (\mathrm{CS} - q_1)/(\mathrm{CS}^* - q_1)$ whenever $\mathrm{CS}^* > q_1$, and the excess time $T - T^*$. Values of NCS above one are possible because a model may repair unreported criteria, which is precisely the generalization from a single report that L2 is designed to expose. The oracle makes speed comparable across tasks that differ in how many criteria initially fail, and it makes explicit that under L2 the reachable acceptance time is bounded by the reporting rate unless the model generalizes.

4.7 Cost and reliability

We record input and output tokens and wall-clock time per run and report CS against tokens as a Pareto frontier rather than a ratio, which is unstable and hides ceilings [15]. For reliability we report the 10th percentile of CS across seeds and the variance components attributable to task, model, and seed.

5 Theoretical Analysis

Proposition 1 (Decomposition)**.** *For $t \ge 2$, $q_t - q_{t-1} = A_t - B_t - D_t$.*

Proof. $w(P_t) = w(P_{t-1}) - w(P_{t-1} \setminus P_t) + w(P_t \setminus P_{t-1})$. Since $\mathcal{S}(s_t) \cap P_t = \emptyset$, the set $P_{t-1} \setminus P_t$ is the disjoint union of its part outside $\mathcal{S}(s_t)$, of weight B_t, and $P_{t-1} \cap \mathcal{S}(s_t)$, of weight D_t. □

Example 1. Let $n = 4$, $w_i = \frac{1}{4}$, $K = 3$, $\theta = \frac{1}{2}$, and no gates. Trajectory A has $P_1 = \{1\}$, $P_2 = \{1, 2\}$, $P_3 = \{1, 2, 3\}$. Trajectory B has $P_1 = \{1\}$, $P_2 = \{2, 3\}$, $P_3 = \{1, 3, 4\}$. Both have $q = (\frac{1}{4}, \frac{1}{2}, \frac{3}{4})$, hence the same $\mathrm{AUC} = \frac{1}{2}$ and $T = T^s = 2$. For A, $\mathrm{RR} = 0$ and $\mathrm{CS} = \frac{1}{2}$. For B, $B_2 = \frac{1}{4}$ and $B_3 = \frac{1}{4}$, so $\mathrm{RR}_2 = 1$, $\mathrm{RR}_3 = \frac{1}{2}$, $\mathrm{RR} = \frac{3}{4}$; moreover $Q_1 = \emptyset$, $Q_2 = \{3\}$, $Q_3 = \{1, 3, 4\}$, so $\mathrm{CS} = \frac{1}{3}$.

Proposition 2 (Insufficiency of quality sequences)**.** *No function of $(q_1, \ldots, q_K)$ alone can distinguish trajectories with different regression behavior. In particular* AUC*, T,* AR@K*, and any statistic of the per-turn increments are blind to regressions.*

Proof. Example 1 gives two trajectories with identical quality sequences but $\mathrm{RR} = 0$ and $\mathrm{RR} = \frac{3}{4}$. Any function of the sequence takes the same value on both. □

Proposition 3 (Properties of the Convergence Score)**.** *(i) $q_t^p \le q_t$, q_t^p is nondecreasing in t, and $q_K^p = q_K$. (ii)* CS $\le$ AUC*, with equality if and only if $P_1 \subseteq P_2 \subseteq \cdots \subseteq P_K$. (iii)* CS $\in [0, 1]$ *and has no free parameter.*

Proof. (i) $Q_t \subseteq P_t$ gives the first claim. $Q_t = \bigcap_{u \ge t} P_u \subseteq \bigcap_{u \ge t+1} P_u = Q_{t+1}$ gives monotonicity, and $Q_K = P_K$. (ii) The inequality follows from (i). Since all weights are positive, $w(Q_t) = w(P_t)$ holds exactly when $Q_t = P_t$, that is, when $P_t \subseteq P_u$ for all $u > t$. Requiring this for every t gives the nested chain. (iii) is immediate. □

Proposition 4 (Feedback efficiency)**.** *Whenever* FE *is defined,* FE ≤ 1*, with equality if and only if $q_K = 1$ and $q_2 = \cdots = q_{K-1} = 1$ (the gap closes entirely in the first feedback round and stays closed). If $q_{t-1} < 1$ for all t, then* FE *equals the headroom-weighted mean of the per-turn gap closures g_t with weights $1 - q_{t-1}$.* FE > 0 *if and only if $q_K > q_1$. If $q_1 \le 1 - \eta$ for some $\eta \in (0, 1)$, then* FE $\ge -q_1/(1 - q_1) \ge -(1 - \eta)/\eta$*, and the same bound holds for the pooled estimate over eligible runs.*

Proof. The $t = 2$ term of the denominator is $1 - q_1$, and all other terms are nonnegative, so $q_K - q_1 \le 1 - q_1 \le \sum_{t \ge 2}(1 - q_{t-1})$. Equality requires $q_K = 1$ and $\sum_{t \ge 3}(1 - q_{t-1}) = 0$. For the weighted mean, $(1 - q_{t-1})g_t = q_t - q_{t-1}$ telescopes to $q_K - q_1$. The sign statement follows because the denominator is positive. For the lower bound, the denominator is at least $1 - q_1 > 0$ and, since $q_K \ge 0$, the numerator is at least $-q_1$, so $\mathrm{FE} \ge -q_1/(1 - q_1)$. This quantity is

Table 1: Hidden specification for one task instance: single-player battle royale against bots in a browser. Weights are in percent. One subsystem is a gate.

Subsystem	Weight	Example criterion
Stability (gate)	15	no crash in 5 min
Movement	12	sprint and jump respond
Camera	10	no jitter or clipping
Physics	12	no fall-through
Combat	12	$\geq$3 weapon types
Zone and flow	12	$\geq$3 shrink phases
Map and loot	8	loot is reachable
Bots	8	bots avoid the zone
Interface	6	accurate minimap
Restart	5	clean restart

decreasing in q_1 on $[0, 1)$ and equals $-(1-\eta)/\eta$ at $q_1 = 1-\eta$. Pooling gives $\sum_i n_i / \sum_i d_i = \sum_i d_i \, \mathrm{FE}_i / \sum_i d_i$ for numerators n_i and positive denominators d_i, a weighted mean of per-run values, which cannot fall below their minimum. □

Remark. Without the eligibility rule, FE has no lower bound: a loss from a near-ceiling state is divided by a small denominator. With it, the bound above applies (-9 at the default $\eta = 0.1$), which is loose in practice. This is why the eligibility rule and ratio-of-sums pooling are part of the definition. Section 7 quantifies the effect.

6 Evaluation Design

6.1 Task families and instances

Tasks span four families: games (for example battle royale against bots, tower defense, racing), applications (task manager, budgeting tool), data tools (CSV explorer, log analyzer), and interactive sites (booking flow, documentation site). Each family includes tiers of roughly 15, 30, and 50 or more criteria. Every task pins down its platform and runtime, player model, explicit non-goals, and deliverable format, because a vague request such as "a stable battle royale game" covers products that differ by orders of magnitude in difficulty. Table 1 shows a worked specification.

Contamination control. Specifications are parameterized (theme, map size, number of zone phases, weapon list) and each instance carries a twist rule, such as a zone that moves instead of only shrinking. A portion of tasks stays private, and releases are versioned and refreshed, in the spirit of contamination-aware code evaluation [14].

6.2 Verification

Criteria are checked programmatically wherever possible: unit and integration tests, headless execution with scripted inputs, crash and error monitors, and performance probes on reference hardware. Subjective criteria receive observable anchors (for example a bound on pixel jitter during a scripted sprint) and are scored by multiple blind judges from a model family different from the one under test [13], with inter-rater agreement reported. Subjective items form a minority of total weight, and results are reported with and without them. Verifier noise would create spurious regressions, so each check is run repeatedly and decided by majority, a regression is counted only if the criterion is confirmed passing on a_{t-1} and confirmed failing on a_t, and the flake rate is reported per criterion.

6.3 Simulator validation

Replacing a human by a fixed policy requires evidence that the substitution is reasonable. We require three checks: blind-rater plausibility of simulator messages, agreement between the simulator's chosen defect and what human testers report first (Cohen's κ [19]), and agreement between model rankings under simulator-guided and human-guided sessions (Kendall's τ_b [20]). The selection rule is itself an experimental factor: humans tend to report the most salient defect, not the highest-weight one, so we ablate priority, salience, and random orderings and report how absolute scores and rankings move.

6.4 Statistical design

The unit of analysis is the task, because runs within a task are correlated through the specification and verifier. Intervals use a cluster bootstrap over tasks with seeds nested within tasks [17], and model comparisons use paired contrasts or a mixed-effects model with random task and task-by-model effects, with Holm correction for pairwise tests [18]. Decoding settings, K, token and time limits, and tool access are declared in advance, and metric definitions, θ, η, δ, and the analysis plan are pre-registered. Sensitivity analyses vary θ, K, criterion weights (for example Dirichlet perturbations of the nominal weights), and the simulator policy, and report rank stability. At least one open-weights model is included because served models can change without notice.

6.5 Planned pilot

A first empirical study would use 12 tasks (four per family), 3 models, 4 levels, $K = 5$ (five attempts and four feedback messages), and 3 seeds, totaling $12 \times 3 \times 4 \times 3 = 432$ runs. The effective sample size for model comparisons is 12, not 432, so the pilot is intended to estimate variance components, verifier reliability, and simulator agreement, and to check whether the ladder produces separable curves, not to rank models.

Table 2: Synthetic profile parameters. f_0: unprompted repair; r_2, c_2: reported and collateral repair at L2; r_3: repair at L3; b: per-criterion breakage; ν: multiplier of f_0 at L1.

Profile	f_0	r_2	c_2	r_3	b	ν
Steady	0.04	0.85	0.08	0.85	0.010	1.5
Churner	0.08	0.95	0.20	0.92	0.070	1.5
Self-driven	0.22	0.70	0.15	0.75	0.020	1.3
Dependent	0.02	0.90	0.04	0.95	0.020	1.2
Weak	0.02	0.12	0.02	0.15	0.030	1.2

7 Synthetic Study

7.1 Setup

We exercise the metrics on stochastic model profiles. The purpose is to show where metrics behave differently and not to say anything about real systems; the parameters are chosen for illustration and are not calibrated to any model. Each task instance has $n = 30$ criteria with weights drawn from a symmetric Dirichlet distribution with concentration 2, and the first attempt passes each criterion independently with probability 0.45. There are 200 task instances, 5 seeds each, $K = 5$, $\theta = 0.70$, no gates, and the first attempt is shared across levels. At each turn a profile acts on the failing criteria as follows. At L0 each is repaired with probability f_0. At L1 each is repaired with probability $\min(1, \nu f_0)$, with $\nu \in \{1.2, 1.3, 1.5\}$ depending on the profile. At L2 the reported criterion (the $\prec$-first failing one) is repaired with probability r_2 and every other failing criterion with probability c_2. At L3 every failing criterion is repaired with probability r_3. Independently, each passing criterion breaks with probability b. Table 2 lists the five profiles. All random draws derive from a single fixed master seed (20261003). Intervals are cluster bootstrap over the 200 tasks (2000 resamples); the largest half-width is 0.008 for AUC, CS, RR, and FE, and 0.03, 0.08, 0.05 for AR@5, RMTA, and the stability gap.

7.2 Results

Conventional summaries and the Convergence Score disagree. Table 3 shows that the Churner profile has the highest AUC (0.649), the highest acceptance rate (0.893), and the shortest restricted mean time to acceptance (3.76). It also breaks roughly seven percent of its passing weight per turn (RR = 0.071), against 0.011 for Steady. Ranked by AUC the order is Churner, Self-driven, Steady; ranked by CS it is Self-driven, Steady, Churner. The stability gap is 0.257 turns for Churner and 0.038 for Steady: Churner reaches acceptance sooner but is far more likely to lose it again. This is the situation Proposition 2 anticipates, and Figure 2(a) shows it directly, since the persistent curve of Churner lies well below its raw curve while Steady's two curves almost coincide.

The ladder and the Self-Sufficiency Ratio. Table 4 and Figure 2(b) report the ladder. The naive ratio $\mathrm{CS}_0/\mathrm{CS}_3$ ranks the Weak profile as the most independent (0.791), above Self-driven (0.746), even though Weak gains nothing from its own review and only 0.083 from L3. The Self-Sufficiency Ratio orders the profiles differently: Self-driven is highest (0.430), Steady is near zero (0.058), and Weak is lowest (-0.343), reflecting that self-revision lowers its persistent quality. The Churner and Dependent profiles also have negative SSR_0, consistent with their b and f_0 values. The ladder is not monotone for Self-driven ($\mathrm{CS}_2 = 0.619 < \mathrm{CS}_1 = 0.650$): in this profile, a single specific report draws effort toward one criterion at the expense of unprompted repair. We include this because it shows concretely why monotonicity is a hypothesis to test.

Behavior of feedback efficiency near the ceiling. We ran the Steady profile at L2 with a first-attempt pass probability of 0.9 (1000 runs, 988 with a defined per-turn mean). Without eligibility, the standard deviation is 0.59 for the mean of per-turn closures and 0.56 for FE, and the minimum is -4.58 and -11.59 respectively. So the headroom-weighted form is not more stable by itself; both have heavy left tails created by losses from near-ceiling states. With the eligibility rule $q_1 \leq 0.9$ (434 runs), the standard deviations are 0.39 and 0.17, the minima -3.15 and -0.20, and the share of runs below -1 is 0.9% and 0. The definition therefore needs the eligibility rule and ratio-of-sums pooling together, as stated in Section 4.

Simulator policy changes absolute scores. Table 5 compares three selection rules at L2 for two profiles. Absolute CS falls by about 0.06 to 0.07 when the highest-weight failing criterion is replaced by a random one, and by about 0.09 to 0.10 for the lowest-weight one, while the order of the two profiles is unchanged in this example. Absolute scores are therefore policy dependent, which supports treating the policy as a reported factor and checking rank stability across policies.

Limits of this study. The profiles are independent Bernoulli processes with no gates, no correlated criteria, and no notion of edit size, and their parameters are chosen by the author. The study demonstrates metric behavior and supports the formal claims above. It is not evidence about how any language model behaves.

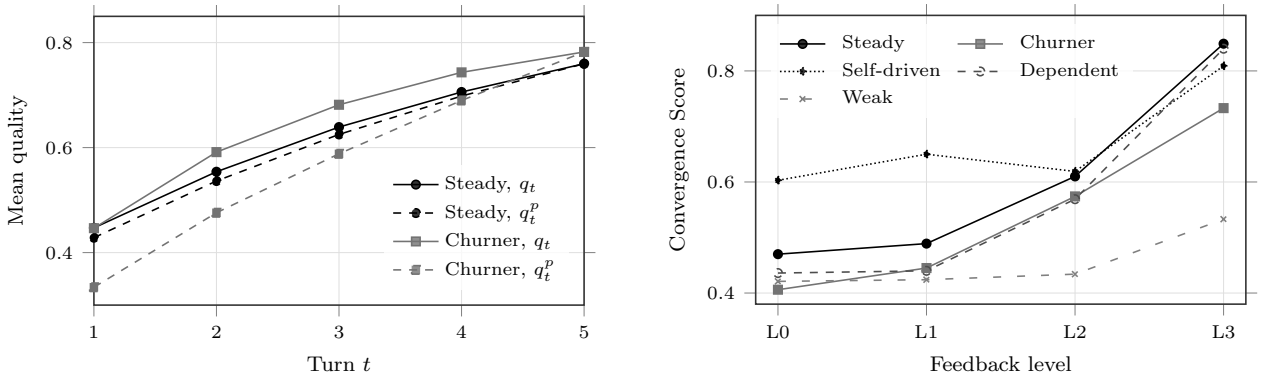
8 Hypotheses and Discussion

The framework is useful only if it can fail. We state four hypotheses for the planned empirical program.

H1. Model rankings by CS at L2 differ from rankings by first-attempt quality on the same tasks (rank correlation clearly below one).

Table 3: Trajectory metrics at L2 (specific feedback). Higher is better except RR, RMTA ($K+1 = 6$ means never accepted), and the stability gap Δ. FE is the pooled ratio of sums over eligible runs ($q_1 \leq 0.9$), as defined in Section 4.

Profile	AUC	CS	RR	FE	AR@5	RMTA	Δ
Steady	0.621	0.610	0.011	0.189	0.772	4.29	0.038
Churner	0.649	0.574	0.071	0.218	0.893	3.76	0.257
Self-driven	0.641	0.619	0.020	0.216	0.861	3.95	0.075
Dependent	0.590	0.569	0.021	0.146	0.560	4.85	0.059
Weak	0.461	0.434	0.030	0.010	0.035	5.89	0.041



(a) Raw and persistent quality at L2

(b) Convergence Score along the feedback ladder

Figure 2: Synthetic study. (a) The persistent curve exposes the instability that the raw curve hides. (b) The ladder separates profiles with similar first attempts, and is not monotone for the Self-driven profile.

H2. RR and CS separate models that AUC and AR@K do not, at least on the hardest tier.

H3. Feedback dependence, as measured by SSR_0, is only weakly related to first-attempt quality.

H4. For most current models, the L0 revision gain is small relative to the L2 and L3 gains, consistent with prior reports that intrinsic self-correction is limited [6, 7].

Results against H1 and H3 would show that convergence is either redundant with existing axes or reducible to them. A strongly monotone ladder and a clearly nonzero SSR_0 for some models would support treating feedback dependence as a distinct property.

On autonomy. In the human-guided workflow that motivates this work, the person supplies the critique and the model executes, which is collaboration and not a general capability. The ladder turns this observation into a measurement: how much of the improvement survives when the critique is removed. Where models fall on that gradient is an empirical result. Any relation to general intelligence is a separate discussion that the benchmark does not presuppose.

Threats to validity. *Construct validity:* a hidden specification is a proxy for user preference, and its weights are subjective, which we address through sensitivity analysis over weights and the visible/hidden split. *Simulator artifacts:* a rigid policy may favor certain behaviors, so validation against humans and policy ablations are required. *Verifier noise:* flaky checks create false regressions, handled by repeated verification and confirmation rules. *Contamination:* parameterized instances and a private split reduce, but do not remove, memorization. *Compute confound:* longer reasoning may buy convergence, so cost-matched Pareto reporting is required. *Finite horizon:* with K small we measure progress over a bounded horizon and not convergence in the limit, so we use the term K-horizon convergence and recommend longer-horizon extensions for oscillation. *Saturation:* strong models may saturate lower tiers, so harder tiers and refreshed tasks are planned from the start. *Drift:* served models change, so identifiers and dates are logged and open-weights baselines are included.

Limitations. The framework addresses software artifacts with automatable checks and does not cover creative taste. One defect per round at L2 is a deliberate simplification, and a multi-defect variant is a natural extension. Engineering the verifiers is the main cost of building tasks. And the empirical claims that matter most remain to be tested.

9 Call for Collaboration

This document specifies a measurement framework and reports no evaluation of language models. Several components of the planned work need expertise or resources that a single author does not have. We list them below, together with the design questions on which criticism would be most useful.

Where contributions are needed.

Table 4: Feedback ladder. q_1 is the shared first-attempt quality. CS_0/CS_3 is the naive autonomy ratio. SSR_0 uses revision gains relative to the best level.

Profile	q_1	CS_0	CS_1	CS_2	CS_3	CS_0/CS_3	SSR_0
Steady	0.447	0.470	0.489	0.610	0.849	0.554	0.058
Churner	0.446	0.406	0.445	0.574	0.733	0.554	−0.140
Self-driven	0.448	0.603	0.650	0.619	0.809	0.746	0.430
Dependent	0.449	0.436	0.440	0.569	0.840	0.519	−0.033
Weak	0.450	0.421	0.424	0.434	0.533	0.791	−0.343

Table 5: CS at L2 under three simulator selection rules. These values come from a separate simulation run, so the Priority column differs from Table 3 by Monte Carlo variation (at most 0.003).

Profile	Priority	Random	Lowest weight
Steady	0.612	0.557	0.527
Dependent	0.572	0.503	0.471

1. *Task and verifier engineering.* Implementing parameterized task instances in the four families of Section 6, with programmatic verifiers, headless execution, and per-criterion flake-rate measurement.
2. *Model access and compute.* Running the pilot of Section 6 (432 runs), including at least one open-weights model, and recording tokens and wall-clock time for the cost analysis.
3. *Human studies.* Testers for the simulator validation: blind plausibility ratings, agreement between the simulator's chosen defect and the defect humans report first, and agreement between model rankings under simulated and human-guided sessions.
4. *Statistical and methodological review.* The mixed-effects specification, the pre-registration, the variance-component estimates the pilot is meant to deliver, and the sensitivity analyses over weights, θ, and K.
5. *Theory and extensions.* Correlated criteria, a multi-defect variant of L2, a notion of edit size, and longer-horizon versions of the Convergence Score.

Open design questions. (i) How criterion weights should be set and how sensitive rankings are to them. (ii) Whether the priority rule for the L2 simulator should be replaced by a salience-based or randomized rule, given the policy dependence in Table 5. (iii) Whether persistence measured to the end of the horizon is the right notion for small K, or whether a windowed variant is preferable. (iv) Whether the ordering $L0 \leq L1 \leq L2 \leq L3$ holds for real models, since the synthetic study already shows a profile for which it does not. (v) Which task families are most likely to saturate at current model capability.

Process. Comments on definitions, task specifications, verifier implementations, or independent replications of the synthetic study can be sent to the address on the first page. Authorship and credit for the pilot are to be agreed before data collection begins and recorded in the pre-registration, and pilot results will be reported whether or not they support H1–H4.

10 Conclusion

LinearBench proposes to measure how language models converge on complex products under controlled feedback: how fast, how stably, and with how much external information. The contribution is a formal framework, a feedback ladder that makes dependence on external guidance a measurable gradient, and a metric suite whose key properties are proved: quality-only summaries are provably blind to regressions, the Convergence Score is parameter free and bounded by the area under the quality curve, and feedback efficiency is bounded above by one. A synthetic study shows concrete disagreements between conventional and proposed summaries. The next steps are the prototype task, the pilot described in Section 6, simulator validation against human sessions, and pre-registration of the analysis plan. Section 9 lists the contributions that would most help with these steps.

References

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

[2] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *ICLR*, 2024.

[3] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. arXiv:2406.12045, 2024.

[4] X. Wang, Z. Wang, J. Liu, Y. Chen, L. Yuan, H. Peng, and H. Ji. MINT: Evaluating LLMs in multi-turn interaction with tools and language feedback. In *ICLR*, 2024.

[5] P. Laban, H. Hayashi, Y. Zhou, and J. Neville. LLMs get lost in multi-turn conversation. arXiv:2505.06120, 2025.

[6] J. Huang, X. Chen, S. Mishra, H. S. Zheng, A. W. Yu, X. Song, and D. Zhou. Large language models cannot self-correct reasoning yet. In *ICLR*, 2024.

[7] R. Kamoi, Y. Zhang, N. Zhang, J. Han, and R. Zhang. When can LLMs actually correct their own mistakes? A critical survey of self-correction of LLMs. *Transactions of the Association for Computational Linguistics*, 2024.

[8] A. Madaan, N. Tandon, P. Gupta, et al. Self-Refine: Iterative refinement with self-feedback. In *NeurIPS*, 2023.

[9] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. In *NeurIPS*, 2023.

[10] J. Yang, A. Prabhakar, K. Narasimhan, and S. Yao. InterCode: Standardizing and benchmarking interactive coding with execution feedback. In *NeurIPS Datasets and Benchmarks Track*, 2023.

[11] X. Chen, M. Lin, N. Schärli, and D. Zhou. Teaching large language models to self-debug. In *ICLR*, 2024.

[12] Z. Gou, Z. Shao, Y. Gong, Y. Shen, Y. Yang, N. Duan, and W. Chen. CRITIC: Large language models can self-correct with tool-interactive critiquing. In *ICLR*, 2024.

[13] L. Zheng et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *NeurIPS Datasets and Benchmarks Track*, 2023.

[14] N. Jain et al. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. arXiv:2403.07974, 2024.

[15] S. Kapoor, B. Stroebl, Z. S. Siegel, N. Nadgir, and A. Narayanan. AI agents that matter. arXiv:2407.01502, 2024.

[16] E. L. Kaplan and P. Meier. Nonparametric estimation from incomplete observations. *Journal of the American Statistical Association*, 53(282):457–481, 1958.

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

[18] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian Journal of Statistics*, 6(2):65–70, 1979.

[19] J. Cohen. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[20] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.