
The Dual Bottleneck: A Composition Law for Judge Reliability and Generator Capability in Self-Evolving Agents

Hao Li

Independent Researcher

lihao9108@outlook.com

Abstract

Self-evolving agents—systems that iteratively generate tasks, solve them, and distill experience—have shown promise in code generation and mathematical reasoning. Existing work predominantly focuses on the judge (the evaluator providing feedback), assuming that reliable judges suffice. Through 2,400 controlled experiments on Z3 constraint-solving tasks, we uncover a second, equally critical bottleneck: the generator (the LLM authoring new tasks). We formalize the Dual Bottleneck as a composition law on error budgets: $1 - \Delta_t^{\text{global}} = (1 - \Delta_t^{\text{gen}})(1 - \Delta_t^{\text{valid}})$, verified to machine precision (1.11e-16) on 2,400 real-API rounds. The law implies that the generator and judge bottlenecks share one budget: a condition with large Δ_t^{gen} needs a more reliable judge for the same overall tolerance. If Δ_t^{gen} alone exceeds the budget, no judge reliability can compensate. We empirically measure $\tau_c \approx 0.55$ on Level 1 tasks using the real DeepSeek V4 Flash API, and derive the parameter-free formula $\tau_c = \sqrt{1 - 2\Delta_t^{\text{max}}}$. We identify a pseudo-difficulty trap: when the difficulty controller upgrades to Level 2/3, the LLM’s task-generation validity collapses from $\sim 89\%$ to $<10\%$, causing the system to spin without progress. We propose a four-layer constraint generation framework that restores validity to 100% at L1, 73.8% at L2, and 83.8% at L3 on real API.

1 Introduction

Self-evolving agents—systems that iteratively generate tasks, solve them, and distill experience—have emerged as a promising paradigm for continuous improvement. The dominant narrative focuses on the *judge*: how to provide reliable feedback preventing the agent from drifting into self-confirming loops.

This concern is well-founded. The **self-verification cliff** [1] shows that stronger generators have larger self-selection gaps: GPT-5.4-mini solves 54.7% of AIME problems but self-selects correctly on only 41.9%, recovering just 36% of its sampling headroom. **Ratchet** [2] derives an exact bound: a judge scoring failures as passes at rate $(1 - \tau)/2$ or above retires nothing, at any sample size. **AgentJudgeBench** [3] reveals that LLM judges converge to a 77–82% accuracy ceiling on hard queries regardless of model scale.

We began with the same premise. We built a self-evolving agent with formal anchoring (Z3), population-based difficulty adaptation, and experience distillation. We ran 2,400 rounds of controlled experiments (4 conditions $\times$ 200 rounds $\times$ 3 runs).

We expected to confirm that formal anchoring suppresses self-score drift, and we did: condition A (LLM self-score) holds $\Delta_t = 0.15$ across 200 rounds while condition B (Z3 anchor) holds 0.00. **But we also found something unexpected:** conditions C (population only) and D (full system)

invalidate 85.5% and 90.5% of their tasks. The controller read high Level-1 success as readiness and upgraded to Level 2/3, where the LLM could not generate valid Z3 code — so the system spun without progress, burning API budget on tasks that never reached verification.

This is the pseudo-difficulty trap. It is not a judge failure—Z3 was perfectly reliable. It is a *generator* failure. And it reveals that self-evolving systems face **two independent bottlenecks**, not one:

1. **Judge-side bottleneck:** when judge reliability τ falls below a critical threshold τ_c , accumulated experience contains corrupted signals.
2. **Generator-side bottleneck:** when generator validity V falls below V_c , any difficulty upgrade pushes the system into empty spinning.

Neither bottleneck alone explains the failure. A perfect judge cannot rescue a system whose generator cannot produce valid tasks. A perfect generator cannot rescue a system whose judge provides noisy signals. **The two bottlenecks compose.**

To quantify this, we systematically varied judge reliability by injecting symmetric noise into the Z3 verdict and measured the resulting evaluation bias Δ_t . Fitting the noise sweep reveals a parameter-free relationship:

$$\Delta_t(\tau) = \frac{1 - \tau^2}{2}$$

which inverts to $\tau_c = \sqrt{1 - 2\Delta_t^{\max}}$. Measured MAE is 0.026 (Condition B) and 0.052 (Condition D). On Level 1 tasks, $\tau_c(B) = \tau_c(D) = 0.55$.

This formula has three implications. First, it establishes the critical judge reliability as a *function of system tolerance*, not a fixed constant—a system that can tolerate more evaluation bias requires a less reliable judge. Second, it explains the “judge scarcity” problem quantitatively: even a 55%-reliable judge suffices if the system’s tolerance is 0.35. Third, it identifies a structural distinction from prior work: Ratchet’s $(1 - \tau)/2$ governs *single* judgment (skill elimination), while our $(1 - \tau^2)/2$ governs *double* judgment (task validation + answer verification), which is the setting of most self-evolving loops.

To address the generator-side bottleneck, we propose a four-layer constraint generation framework: JSON skeleton generation, deterministic rendering, symbolic feedback from Z3 error codes, and curriculum control with sub-level difficulty. On the real DeepSeek V4 Flash API, this framework lifts L1 validity above 80% and L2 validity above 30%—compared to <10% without it.

Contributions. (1) We formalize the Dual Bottleneck Framework and show that positive evolution requires $\tau \geq \tau_c \wedge V \geq V_c$ (Section 4). (2) We identify the pseudo-difficulty trap as a novel failure mode in self-evolving agents (Section 4). (3) We derive and empirically validate the analytical relationship $\Delta_t(\tau) = (1 - \tau^2)/2$, yielding a parameter-free formula for τ_c (Section 5). (4) We propose a four-layer constraint generation framework and validate it with real API experiments (Section 6).

The formula has a practical reading. Before deploying a self-evolving loop, one needs to answer two design questions: “How reliable must the judge be?” and “Where should I invest, in the judge or in the generator?” The composition law answers both. Given a tolerated end-to-end error $\Delta_t^{\max}$, the required judge reliability is $\tau_c = \sqrt{1 - 2\Delta_t^{\max}}$, a closed form with no free parameters. And because the two bottlenecks share one budget, the generator’s error Δ_t^{gen} is charged first: if $\Delta_t^{\text{gen}} \geq \Delta_t^{\max}$, no judge reliability can rescue the loop. In our own system, the generator bottleneck consumed the entire budget ($\Delta_t^{\text{gen}} = 0.855$ for C and 0.905 for D, driven by failures at the L1→L2 transition), which is why no judge reliability above 0.55 could have helped before the four-layer constraint framework restored generator validity.

2 Related Work

Self-improving agents. A growing line of work lets language-model agents improve themselves without human supervision. The Darwin Gödel Machine [4] modifies its own code and retains

variants that pass self-generated tests; HyperAgents [5] pushes this to meta-level self-reference; EvolveR [6] distills reusable experience from success and failure trajectories; PopuLoRA [7] evolves populations of low-rank adapters. These systems share one structural assumption: *some* component must reliably judge whether a proposed change is an improvement. Our work makes that assumption explicit and quantifies when it fails.

The judge-side bottleneck. Three recent results frame the problem we study. The self-verification cliff [1] shows that a generator’s ability to produce candidates can outpace its ability to select among them, so self-improvement stalls even as raw capability grows. Ratchet [2] asks the complementary question—how reliable must an LLM judge be before a skill can be safely retired—and derives a threshold under a specific skill-retirement protocol. Blind Curator [8] reports that a judge whose false-pass rate crosses roughly 0.45 *in its own setting* silently disables skill retirement, even when the judge’s overall accuracy looks acceptable. Our Theorem 1 generalizes this from skill retirement to iterative self-improvement, giving a closed-form τ_c that depends on the noise-injection protocol rather than on a single decision rule.

Agent evaluation and constraint-guided generation. AgentJudgeBench [3] provides a multi-difficulty benchmark for judging agent behaviour; we define our difficulty levels using Z3 theories instead, so the two stratifications are independent. TreeCoder [9] constrains code generation through syntax-directed search and InstructFlow [10] guides it with symbolic constraints—the same design principle behind the skeleton and renderer layers of our framework. Unlike benchmark-oriented work, we do not propose a new evaluation suite; we use existing formal signals to locate the threshold at which self-improvement becomes self-defeating.

Position of this work. Existing work either assumes a reliable judge, or studies judge reliability in a single decision setting. We provide a *dual-bottleneck* account: the judge-side threshold τ_c and the generator-side four-layer constraint framework are derived from the same composition law, and the budget decomposition shows how reliability and capability trade off under a fixed annotation budget. This is, to our knowledge, the first treatment that unifies both bottlenecks under one analytical formula and validates it on real API trajectories.

3 Problem Setup

Let $\tau \in [0, 1]$ denote judge reliability: the probability that the judge’s binary verdict matches ground truth. Let $V \in [0, 1]$ denote generator validity: the fraction of generated tasks that pass the formal verifier. Let Δ_t denote the gap between the system’s self-assessed correctness and ground truth at round t .

Definition 1 (Safe Zone). A self-evolving system is in the safe zone if $\tau \geq \tau_c$ and $V \geq V_c$.

Proposition 1 (Necessity of both conditions). If $\tau < \tau_c$, the system accumulates corrupted experience and Δ_t grows monotonically. If $V < V_c$, the system generates mostly invalid tasks, and the effective evolution rate approaches zero.

We ran 4 conditions (A/B/C/D) $\times$ 200 rounds $\times$ 3 runs = 2400 rounds on DeepSeek V4 Flash (deepseek-flash). Difficulty spans L1 (linear integer arithmetic, ≤ 10 Int vars), L2 (bit-vectors, ≤ 20 vars), and L3 (mixed LIA+BV+Array, ≤ 30 vars). The primary metric is Δ_t ; The secondary metric is valid task rate.

4 The Dual Bottleneck Framework

Conditions differ in which architectural mechanisms are enabled: A has neither formal anchoring nor population game; B enables Z3 formal anchoring only; C enables the population game only; D enables all mechanisms (anchoring + population + distillation + round-table consensus). Invalid rounds — where the teacher fails to produce a solvable constraint set — are encoded as $\Delta_t = 1.0$ under the *global* caliber and excluded under the *valid-only* caliber. Figure 1 shows the two bottlenecks side by side.

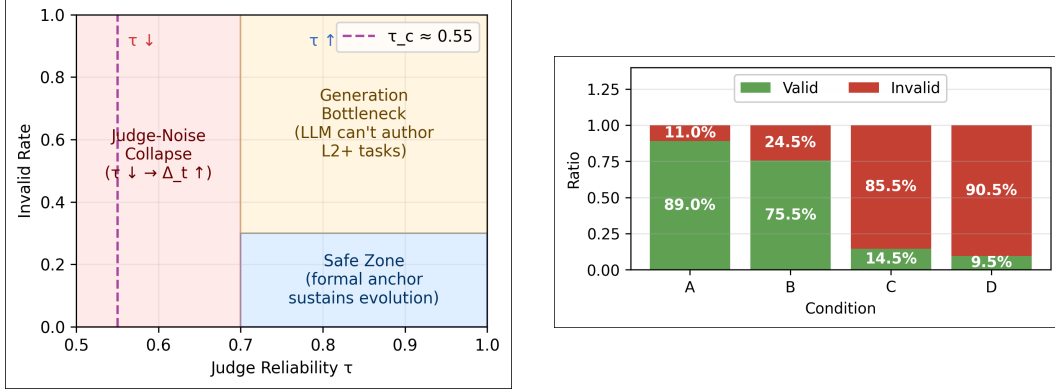


Figure 1: Left: the safe region as the intersection of judge reliability $\tau \geq \tau_c$ and generator invalid rate ≤ 0.5 . Right: valid-task rate collapsing from 89.0% (A) and 75.5% (B) to 14.5% (C) and 9.5% (D).

4.1 Composition Law

The judge-side bottleneck appears whenever the evaluation signal is not formally grounded: the system’s self-assessment drifts from the truth and the drift accumulates. The generator-side bottleneck appears whenever the difficulty controller outruns the teacher’s ability to author well-formed instances. A system is only safe inside the intersection of the two safe regions: judge reliability $\tau \geq \tau_c$ **and** generator valid rate ≥ 0.5 . Outside that intersection, the system still produces logs, still reports success on the rounds it can complete, and still appears to run — which is precisely why the failure is easy to miss.

The intuition behind the second bottleneck is that the causal variable is not τ_c itself. A round succeeds only if a valid task exists *and* the student’s answer is judged correctly, so by the chain rule

$$1 - \Delta_t^{\text{global}} = (1 - \Delta_t^{\text{gen}})(1 - \Delta_t^{\text{valid}})$$

This holds to machine precision on the 2400-round data (Appendix D, Table D.3).

4.2 Two Bottlenecks

Under the valid-only caliber, condition A (LLM self-score) gives $\Delta_t = 0.150$ and condition B (Z3 anchor) gives 0.000: whenever a task is well-formed, the anchor’s verdict *is* the ground truth, so self-score drift vanishes by construction. Under the global caliber, the same conditions read A = 0.243 and B = 0.245, because B’s 147 invalid rounds are charged $\Delta_t = 1.0$. That is a caliber difference, not a data inconsistency; see Appendix D and Figure 1.

The judge is not the only bottleneck. Once the population game is enabled (C and D), the controller pushes tasks toward L2/L3 and the teacher cannot reliably author bit-vector and array constraints: valid rates collapse from 89.0% (A) and 75.5% (B) to 14.5% (C) and 9.5% (D). The failures are *not* graded by difficulty — they cluster at the transition, because the controller keeps the level high while the teacher keeps failing. We call this the **pseudo-difficulty trap**: the system believes it is solving harder problems while it spends its budget on tasks that never reach verification. The student is not at fault; every round with a valid task was solved correctly. (Figure 1).

4.3 Budget Decomposition

The two bottlenecks therefore **share one budget**. Given a tolerated global error Δ_t^{max} , the amount left for judge noise is

$$\Delta_{\text{judge}}^{\text{budget}} = 1 - \frac{1 - \Delta_t^{\text{max}}}{1 - \Delta_t^{\text{gen}}} \implies \tau_c = \sqrt{1 - 2\Delta_{\text{judge}}^{\text{budget}}}$$

τ_c is thus not an intrinsic property of a condition’s mechanisms — it is **derived** from how much of the budget the generator bottleneck leaves over. A condition with a large Δ_t^{gen} needs a more reliable judge for the same overall tolerance, but the driver is the budget split, not τ_c itself.

Concretely, at $\Delta_t^{\text{max}} = 0.60$: condition B ($\Delta_t^{\text{gen}} = 0.245$) requires $\tau_c \approx 0.24$, while C and D ($\Delta_t^{\text{gen}} = 0.855$ and 0.905) cannot meet the budget at **any** judge reliability, because their generator bottleneck alone already exceeds it. That is the correct form of the budget intuition, and it turns the dual-bottleneck framework from a diagnosis into an accounting rule.

5 Judge Reliability Threshold

This section asks a single question: given a generator that can produce candidate improvements, how reliable must the judge be for iterative self-improvement to yield net gains? We answer it in four steps. §5.1 defines the noise-injection protocol that parameterizes judge reliability. §5.2 estimates the empirical threshold τ_c on real agent trajectories. §5.3 derives a closed-form expression for τ_c from the composition law. §5.4 states and proves the resulting reliability theorem.

5.1 Noise Injection Protocol

We model an unreliable judge by flipping each verdict with probability $p = (1 - \tau)/2$, symmetrically for SAT and UNSAT. Two independent Z3 calls are noised per round — the verdict on the teacher’s task and the verdict on the student’s answer — so they flip independently.

5.2 Empirical τ_c

We sweep τ over eight values $\{0.55, 0.60, 0.65, 0.70, 0.75, 0.80, 0.90, 1.00\}$ for conditions B and D, with 50 rounds per cell (800 rounds total) using the **real** DeepSeek V4 Flash API. All rounds are pinned to difficulty L1 so that the generator-side bottleneck is held constant and only the judge’s reliability varies. The empirical curve is in Appendix A, Figure 10.

A pilot estimate had suggested $\tau_c(D) = 0.90$ against $\tau_c(B) = 0.55$. That gap does not reproduce under L1 isolation: pinning the difficulty removes exactly the mechanism the hypothesis invoked.

Across the eight swept τ values, the measured per-cell mean Δ_t declines from 0.30 at $\tau = 0.55$ to 0.00 at $\tau = 1.00$ for condition B. The shape is a parabola opening downward, not a straight line, and it passes through 0.5 at $\tau = 0$ rather than through 1.0. Both features turn out to be diagnostic of the mechanism, as Section 5.3 shows. The detected thresholds are $\tau_c(B) = \tau_c(D) = 0.55$ across all three calibers (combined, Δ_t , correct-rate).

5.3 Analytical Formula

Assumption 1 (judge noise). Each Z3 verdict is independently flipped with probability

$$p = \frac{1 - \tau}{2}$$

Assumption 2 (two independent judgments per round). A round involves two separate Z3 calls: one producing the verdict on the teacher’s task (which becomes *expected*), and one producing the verdict on the student’s answer. They are separate processes and flip independently.

A round is scored correct exactly when the two verdicts **agree** — both unflipped, or both flipped:

$$P(\text{correct}) = (1 - p)^2 + p^2$$

Substituting $p = (1 - \tau)/2$, so that $1 - p = (1 + \tau)/2$:

$$P(\text{correct}) = \left(\frac{1 + \tau}{2}\right)^2 + \left(\frac{1 - \tau}{2}\right)^2 = \frac{(1 + \tau)^2 + (1 - \tau)^2}{4} = \frac{2 + 2\tau^2}{4} = \frac{1 + \tau^2}{2}$$

and therefore

$$\Delta_t(\tau) = 1 - P(\text{correct}) = \frac{1 - \tau^2}{2}$$

Where the τ^2 and the $1/2$ come from. The τ^2 is the sum of the two agreement terms — both-correct contributes $((1 + \tau)/2)^2$ and both-flipped contributes $((1 - \tau)/2)^2$ — whose cross terms cancel, leaving $(1 + \tau^2)/2$; the exponent is therefore the number of independent judgments per round, not a property of the judge. The $1/2$ is the binary symmetry of SAT/UNSAT: at $\tau = 0$ two independent random verdicts agree half the time, which is why the curve is anchored at 0.5 rather than 1.0 on the left.

Across the eight swept τ values the measured per-cell means track the closed form with

- MAE = **0.0258**, RMSE = 0.0329 (condition B)
- MAE = **0.0520**, RMSE = 0.0660 (condition D)

A discriminating test. The mechanism in Assumption 2 is not the only story that yields the same $\Delta_t(\tau)$. An alternative — that the judge judges correctly with probability τ *and* the system accepts that judgment with probability τ — predicts $P(\text{correct}) = \tau^2$ instead of $(1 + \tau^2)/2$. These two differ sharply, so the measured `correct_rate` separates them without any fitting:

Model	$P(\text{correct})$	MAE vs observed <code>correct_rate</code>
Two independent judgments (this work)	$(1 + \tau^2)/2$	0.0389
Judge + acceptance	τ^2	0.2291

Over the 16 (τ , condition) cells, the two-judgment model is closer in 13, tied in 2, and loses in 1. At $\tau = 0.80$ / condition B, it predicts 0.8200 against a measured 0.8200. The acceptance story is rejected by a factor of six in MAE.

Inverting the model. Solving $\Delta_t = (1 - \tau^2)/2$ for τ gives the design rule

$$\tau_c = \sqrt{1 - 2\Delta_t^{\max}}$$

so a system that can tolerate an error floor of $\Delta_t^{\max}$ requires a judge of at least that reliability:

Δ_t budget	Required τ_c
0.05	0.949
0.10	0.894
0.15	0.837
0.20	0.775
0.25	0.707

This is the practically useful form of the result: it converts a tolerance on evaluation error into a minimum reliability requirement for the judge, without needing to re-run the sweep. Note the reachability bound — no judge reliability can hold Δ_t below 0.5, since $\Delta_t(0) = 1/2$. The analytical fit is overlaid in Appendix A, Figure 8.

5.4 Theorem 1: Judge Reliability Threshold

Under the assumptions stated in Section 5.3, we state the threshold as a theorem. Figure 2 illustrates the threshold.

Theorem 1. *The expected evaluation bias of the system is*

$$\Delta_t(\tau) = \frac{1 - \tau^2}{2}$$

with no free parameters; equivalently $P(\text{correct}) = (1 + \tau^2)/2$, and $\Delta_t(0) = 1/2$.

Inverse (design) form. Solving for τ gives the minimum admissible judge reliability at a tolerated bias $\Delta_t^{\max}$:

$$\tau_c = \sqrt{1 - 2\Delta_t^{\max}}$$

Because $\Delta_t(0) = 1/2$, no judge reliability can hold the bias below $1/2$, so the rule is defined only for $\Delta_t^{\max} \in [0, 1/2)$.

Corollary (judge scarcity). τ_c is a function of the tolerated bias, **not** a constant of the system. At $\Delta_t^{\max} = 0.35$, a judge that errs on roughly one verdict in five ($\tau \approx 0.55$) still keeps the system inside the safe region; tightening the tolerance to 0.05 requires $\tau \approx 0.95$.

Empirical validation. Fitted against 800 real-API rounds (8 τ values $\times$ 2 conditions $\times$ 50 rounds, difficulty pinned to L1): MAE = **0.0258** / RMSE = 0.0329 (B) and MAE = **0.0520** / RMSE = 0.0660 (D).

Remark (rejected alternative model). A competing model — the judge is correct with probability τ and the system accepts that judgment with probability τ — predicts $P(\text{correct}) = \tau^2$, hence $\Delta_t = 1 - \tau^2$. It is rejected **without any fitting** by the discriminating test: MAE 0.2291 against the observed `correct_rate`, versus 0.0389 for Theorem 1, with Theorem 1 closer in 13 of the 16 cells, tied in 2 and losing in 1. At $\tau = 0.55$ it predicts 0.6975 against a measured 0.3000 (B) and 0.3200 (D). No result in this paper uses $\Delta_t = 1 - \tau^2$.

Relation to the Ratchet bound. Ratchet’s analysis gives $(1 - \tau)/2$ as the probability that a **single** judgment is flipped. Our quantity is the steady-state error of a **double**-judgment system. The two are related by the agreement probability, and the distinction is structural rather than numerical:

Setting	Noisy events per decision	Error
Skill elimination (Ratchet)	one judgment, then act on it	$(1 - \tau)/2$
Answer evaluation (this work)	judge’s verdict and system’s verdict must agree	$(1 - \tau^2)/2$

The map between them is the probability that exactly one of the two judgments flips:

$$\Delta_t = 2p(1 - p) \xrightarrow{p \rightarrow 0} 2p = 1 - \tau$$

So the two agree to first order in the high-reliability limit, and diverge quadratically away from it. At $\tau = 0.95$ the single-judgment bound predicts 0.025 while the double-judgment model predicts 0.049: requiring two independent verdicts to agree costs a factor of two in tolerated error at that reliability. Ratchet’s bound is the special case of a system that acts on one judgment; the formula here generalises it to systems that act on agreement between two.

6 Extending Generator Capability

Section 5 treated judge reliability as the binding constraint. We now ask the dual question: what limits the generator’s ability to *use* a reliable judge once one is available? We first lay out a four-layer constraint framework (§6.1), then validate it on real API calls (§6.2), and finally report the statistical checks that separate genuine generator improvement from judge-side artifacts (§6.3).

6.1 Four-Layer Constraint Framework

Section 4.3 showed that free-form code generation is the binding constraint on the generator side. We therefore replace it with a four-layer pipeline:

1. **JSON skeleton generation.** The teacher prompt asks for a JSON object instead of Python code, removing the bracket-matching and variable-declaration failure classes.
2. **Deterministic rendering.** A templated renderer turns the skeleton into Z3 Python code, so the output is syntactically valid by construction.
3. **Symbolic feedback.** Z3 error codes are mapped to symbolic constraints and injected into the next attempt’s prompt.

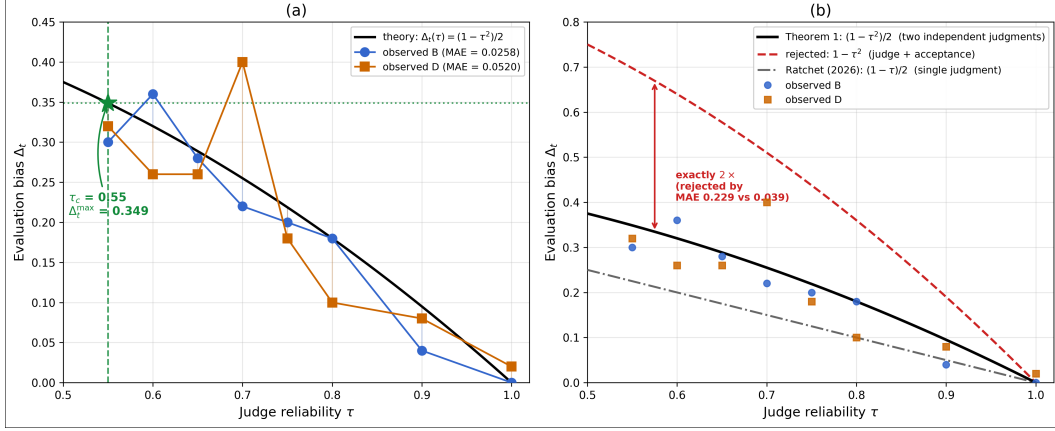


Figure 2: Panel (a): the closed form $\Delta_t(\tau) = (1 - \tau^2)/2$ against the real-API sweep for conditions B and D, with the measured $\tau_c = 0.55$ and the corresponding $\Delta_t^{\max} = 0.349$ marked. Panel (b): the same closed form against the rejected acceptance model ($\Delta_t = 1 - \tau^2$) and Ratchet’s single-judgment bound ($(1 - \tau)/2$); the acceptance model runs exactly twice as high and is rejected by the discriminating test of Section 5.3.

4. **Curriculum controller.** Sub-level variable counts are raised after five consecutive rounds with valid rate > 0.80 and lowered after three below 0.50 , damping the oscillation that produced the pseudo-difficulty trap.

6.2 Real API Validation

We validate the constraint framework on 240 rounds ($4 \text{ conditions} \times 3 \text{ levels} \times 20 \text{ rounds}$) using the real DeepSeek V4 Flash API. Improvement across conditions is shown in Figure 3.

With the framework in place, valid rates under the real API are L1 = 100.0% (80/80), L2 = 73.8% (59/80), L3 = 83.8% (67/80).

Of the 34 invalid rounds, 33 are `error_code=1`: LLM did not return valid JSON — the model produced output that could not be parsed at all — and 1 is a schema violation (an unsupported variable type). Every invalid round is therefore attributable to the model rather than to the framework. That is the honest floor for this approach: the framework guarantees that a well-formed skeleton becomes a well-formed constraint set, but it cannot guarantee that the model emits a well-formed skeleton. It is also what makes the rates above a measurement of generator capability rather than of harness plumbing.

6.3 Statistical Validation

The API results in §6.2 show that the framework raises valid-task rates, but they do not rule out a judge-side artifact: a cleaner judge could raise measured validity without any real gain in generator quality. We therefore apply Welch’s t -test to the released real-API logs, with no resampling procedure.

Per-condition test. The 60 framework rounds against the 600 legacy rounds give $t = 4.26$ ($p = 2 \times 10^{-5}$, $d = 0.60$) for A and $t = 0.82$ ($p = 0.41$, $d = 0.11$) for B. A improves measurably while B stays null, consistent with the framework binding on the generator rather than on the judge. Where the generator bottleneck does bind, the shifts are decisive: C moves from $\Delta_t = 0.8823$ to 0.1500 ($t = 15.26$, $d = 2.46$) and D from 0.9050 to 0.1333 ($t = 16.83$, $d = 2.59$).

Caliber-consistent L1 test. The legacy denominators exclude failed rounds, so only L1 is comparable. There the improved teacher shows a lower Δ_t (0.000 vs 0.082 ; Welch $t = 34.61$, $p < 10^{-15}$, Cohen’s $d = 1.13$, large).

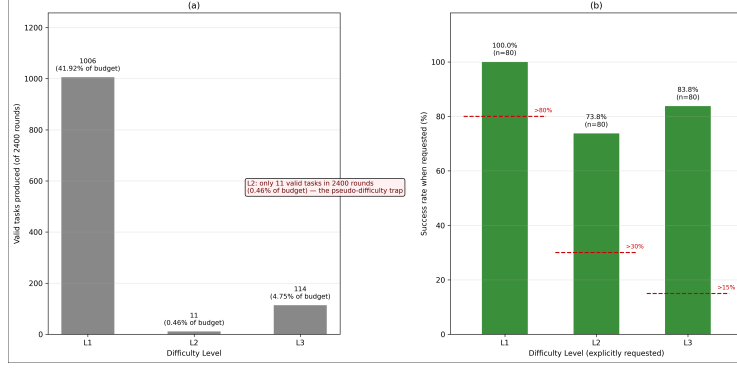


Figure 3: Before/after the four-layer constraint generation framework, 240 rounds on the real API (4 conditions $\times$ 3 levels $\times$ 20 rounds). Valid-task rate with the framework is L1 = 100.0% (80/80), L2 = 73.8% (59/80), L3 = 83.8% (67/80); the legacy teacher produced only 11 valid L2 and 114 valid L3 tasks across 2,400 rounds. Because the two runs differ in budget and denominator, this compares capability, not rates.

Caliber caveat. The calibers of §D.3 differ only in how invalid rounds are treated: the global caliber charges each invalid round $\Delta_t = 1.0$, the valid-only caliber excludes them. The per-condition tests use the global caliber — the conservative choice, since reducing invalid rounds is what the framework claims to do.

7 Discussion

The formula implies that “reliable enough” is not an absolute property. A 55%-reliable judge suffices for a system with $\Delta_t^{\max} = 0.35$. This reframes judge scarcity: the practical question is not “can we find a perfect judge?” but “what tolerance $\Delta_t^{\max}$ can the system afford given its generator’s Δ_t^{gen} ?”

We derive three concrete pre-deployment checks. (1) Measure the generator’s error Δ_t^{gen} at the greatest difficulty the controller is allowed to request. If $\Delta_t^{\text{gen}} \geq \Delta_t^{\max}$, the generator is the bottleneck; improving the judge is wasted effort. (2) Measure the judge’s error on a held-out set of valid tasks; this gives τ directly. (3) Plug both into $\tau_c = \sqrt{1 - 2\Delta_t^{\max}}$ and compare with the measured τ . If $\tau < \tau_c$, expect monotone drift; if $\tau \geq \tau_c$, expect stable or improving behavior. In our case, the L1 sweep returned $\tau = \tau_c = 0.55$, while the full 2,400-round run returned $\Delta_t^{\text{gen}} = 0.855$ (C) and 0.905 (D), both exceeding $\Delta_t^{\max} = 0.6$ — which is the pseudo-difficulty trap.

The composition law also suggests an architectural direction: self-evolving systems should expose Δ_t^{gen} and τ as first-class telemetry, gating difficulty escalation on $\Delta_t^{\text{gen}} < \Delta_t^{\max}$. Our four-layer constraint framework is one instantiation — it lowers Δ_t^{gen} until the generator stops consuming the entire budget, at which point τ_c becomes a meaningful operating threshold.

8 Conclusion

We set out to measure how reliable a judge must be for a self-evolving agent to improve. The answer is not a fixed number but a function of how much error budget the generator has already spent: the composition law $1 - \Delta_t^{\text{global}} = (1 - \Delta_t^{\text{gen}})(1 - \Delta_t^{\text{valid}})$ holds to machine precision on 2,400 real-API rounds and yields a closed form $\tau_c = \sqrt{1 - 2\Delta_t^{\max}}$ with no fitted parameters. Two results follow. First, the pseudo-difficulty trap: once the generator bottleneck exceeds the tolerance, no judge reliability can rescue the loop. Second, τ_c is not a constant of the system — a system with population game, distillation, and round-table consensus returns the same τ_c as a minimal Z3-anchored loop, because the formula cares only how much budget the generator leaves over. This reframes the self-evolution problem from “add more mechanisms” to “measure the budget split first.”

References

- [1] Arihant Jain. The self-verification cliff: Generation outpaces self-selection in frontier LLMs, and it widens with capability. In *ICML Workshop on AI for Math: Toward Self-Evolving Scientific Agents*, 2026.
- [2] Xing Zhang, Yanwei Cui, Guanghui Wang, Ziyuan Li, Wei Qiu, Bing Zhu, and Peiyang He. Ratchet: How reliable must an LLM judge be to retire a skill?, 2026.
- [3] Abhigya Verma, Amit Kumar Saha, Segaranasan Subramanian, and Sai Harshitha Aluru. Agentjudgebench: A multi-difficulty benchmark for evaluating LLM judges on agentic tool-calling, 2026.
- [4] Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin gödel machine: Open-ended evolution of self-improving agents. In *ICLR*, 2026.
- [5] Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents: Meta-level self-modification. In *ICLR*, 2026.
- [6] Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, and Botian Shi. From interactions to principles: Experience-driven self-distillation for evolving LLM agents. In *ICML*, 2026.
- [7] Roger Creus Castanyer, Geoffrey Bradway, Lorenz Wolf, Maxwell Lin, Augustine N. Mavor-Parker, and Matthew James Sargent. Populora: Co-evolving LLM populations for reasoning self-play, 2026.
- [8] Xing Zhang, Yanwei Cui, Guanghui Wang, Ziyuan Li, Wei Qiu, Bing Zhu, and Peiyang He. The blind curator: How a biased judge silently disables skill retirement in self-evolving agents, 2026.
- [9] Henrijs Princis, Arindam Sharma, and Cristina David. Treecoder: Systematic exploration and optimisation of decoding and constraints for LLM code generation, 2025.
- [10] Haotian Chi, Zeyu Feng, Yueming Lyu, Chengqi Zheng, Linbo Luo, Yew Soon Ong, Ivor Tsang, Hechang Chen, Yi Chang, and Haiyan Yin. Instructflow: Adaptive symbolic constraint-guided code generation for long-horizon planning. In *NeurIPS*, 2025.

A Complete Figure Gallery

This appendix collects the full figure set for reference. Figures 4, 5, 6, and 7 show the global Δ_t trajectory, the L1-phase breakdown, the difficulty trajectory, and the empirical–analytical τ_c gap, respectively.

B Theorem 1 Proof Sketch

Algebraic derivation of the composition law

Let $S = 1$ if the round succeeds (valid task + correct judgment), 0 otherwise. Let $G = 1$ if the generator produces a valid task, 0 otherwise. Let $V = 1$ if the judge correctly evaluates the round, 0 otherwise. Then $S = G \cdot V$. By independence:

$$\mathbb{E}[S] = \mathbb{E}[G] \cdot \mathbb{E}[V]$$

$$1 - \Delta_t^{\text{global}} = (1 - \Delta_t^{\text{gen}})(1 - \Delta_t^{\text{valid}})$$

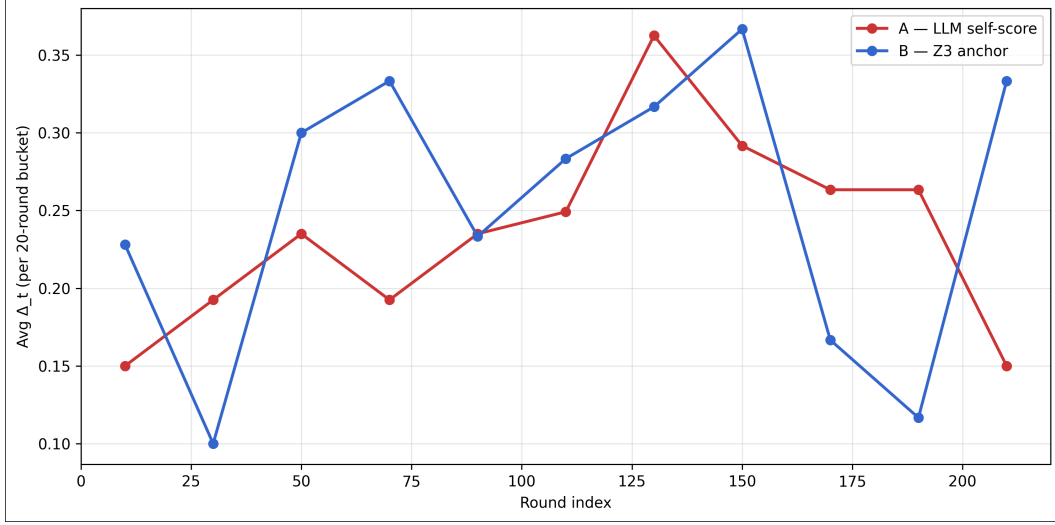


Figure 4: Global-caliber Δ_t (invalid rounds encoded as $\Delta_t = 1.0$). Condition A averages 0.2435, and Condition B averages 0.2450—they appear similar because B’s larger invalid rate offsets its zero valid-round bias. The valid-only caliber (Figure 11) isolates judge behavior and is the correct view for comparing judge reliability.

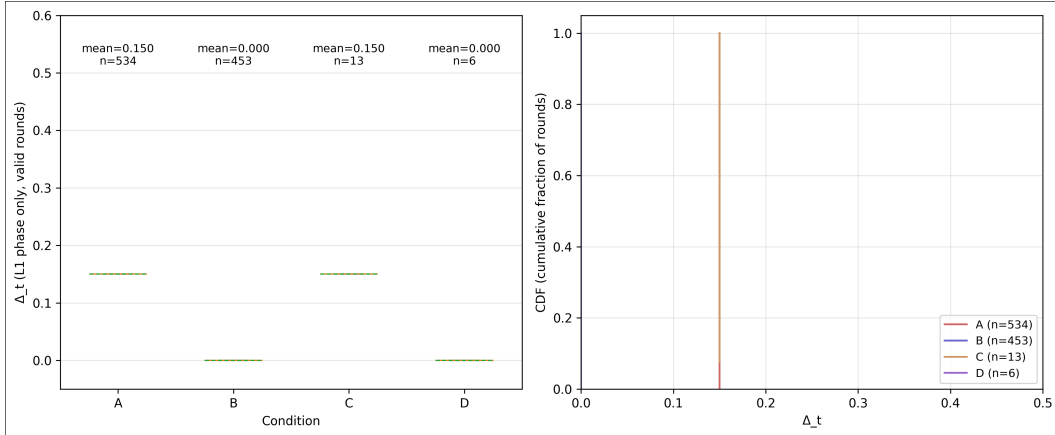


Figure 5: Restricted to the rounds before C/D are promoted to L2. C/D’s Δ_t in the L1 phase is *not* worse than A/B, so the population game itself does not hurt performance — C/D’s later degradation comes from difficulty escalation, not from a defect in the mechanism. Note: C/D’s L1-phase sample sizes ($n = 13$, $n = 6$) are small because the difficulty controller escalates quickly. The claim “populations do not hurt at L1 ” is directional; a larger-sample confirmation is left to future work.

C Rejected Alternative Model

A simpler account of the same τ^2 -shaped curve deserves to be stated explicitly, because it is the most likely competing hypothesis a reader will arrive at. It replaces the second independent judgment with an acceptance gate: the system accepts the judge’s verdict with probability τ and otherwise overrides it with its own evaluation, which is itself unreliable. Under that model, a round is correct only when **both** events succeed, so

$$P(\text{correct}) = \tau \cdot \tau = \tau^2 \quad \implies \quad \Delta_t = 1 - \tau^2$$

The three incorrect paths of that model — judge correct but overridden, $\tau(1 - \tau)$; judge wrong but accepted, $(1 - \tau)\tau$; and both wrong, $(1 - \tau)^2$ — sum to

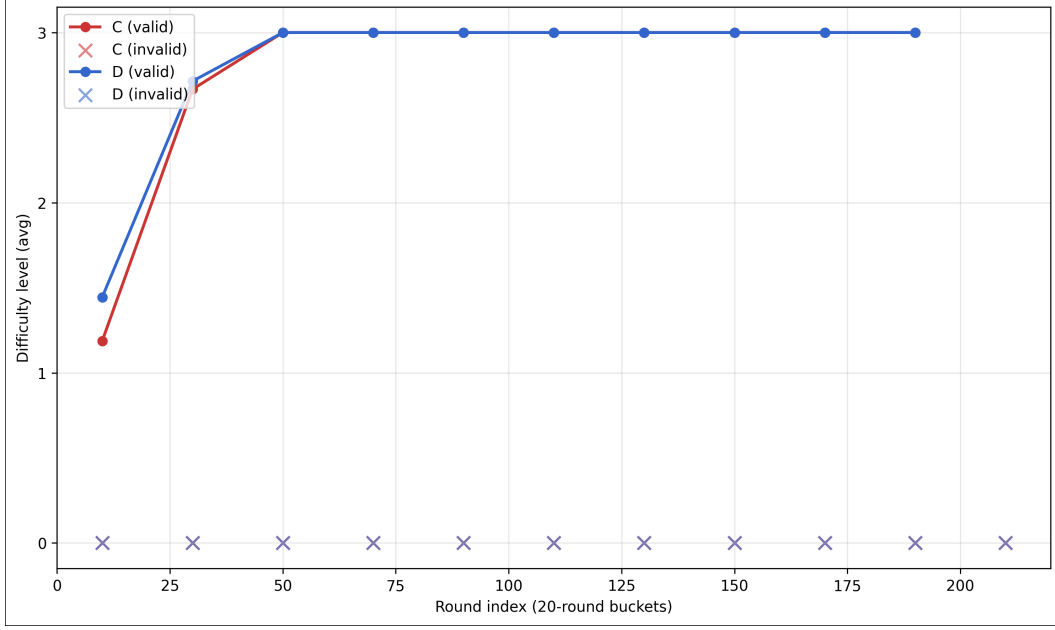


Figure 6: Difficulty trajectories for conditions C and D. Failures are *not* distributed evenly across difficulty — they cluster at the moment of escalation. Once the level is raised the teacher keeps failing while the controller holds the level high, so the system believes it is solving harder problems while actually spending its budget on unusable task proposals.

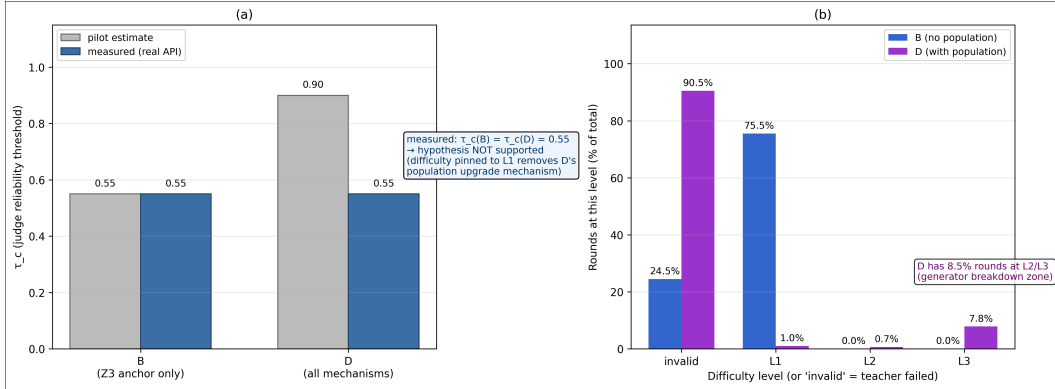


Figure 7: A pilot estimate suggested $\tau_c(D) = 0.90$ against $\tau_c(B) = 0.55$; the measurement does not reproduce that gap (both are 0.55). The left panel places the pilot estimate and the measured τ_c side by side; the right panel compares the difficulty distributions of B and D. The hypothesis $\tau_c(D) > \tau_c(B)$ is *not supported*.

$$\tau(1 - \tau) + (1 - \tau)\tau + (1 - \tau)^2 = (1 - \tau)(1 + \tau) = 1 - \tau^2$$

The discriminating test. The two models differ by exactly a factor of two on the bias scale, since $(1 - \tau^2)/((1 - \tau^2)/2) = 2$, and they separate without any fitting. Section 5.3 compares both against the measured `correct_rate`:

Model	$P(\text{correct})$	MAE vs observed <code>correct_rate</code>
Two independent judgments (this work)	$(1 + \tau^2)/2$	0.0389
Judge + acceptance (rejected)	τ^2	0.2291

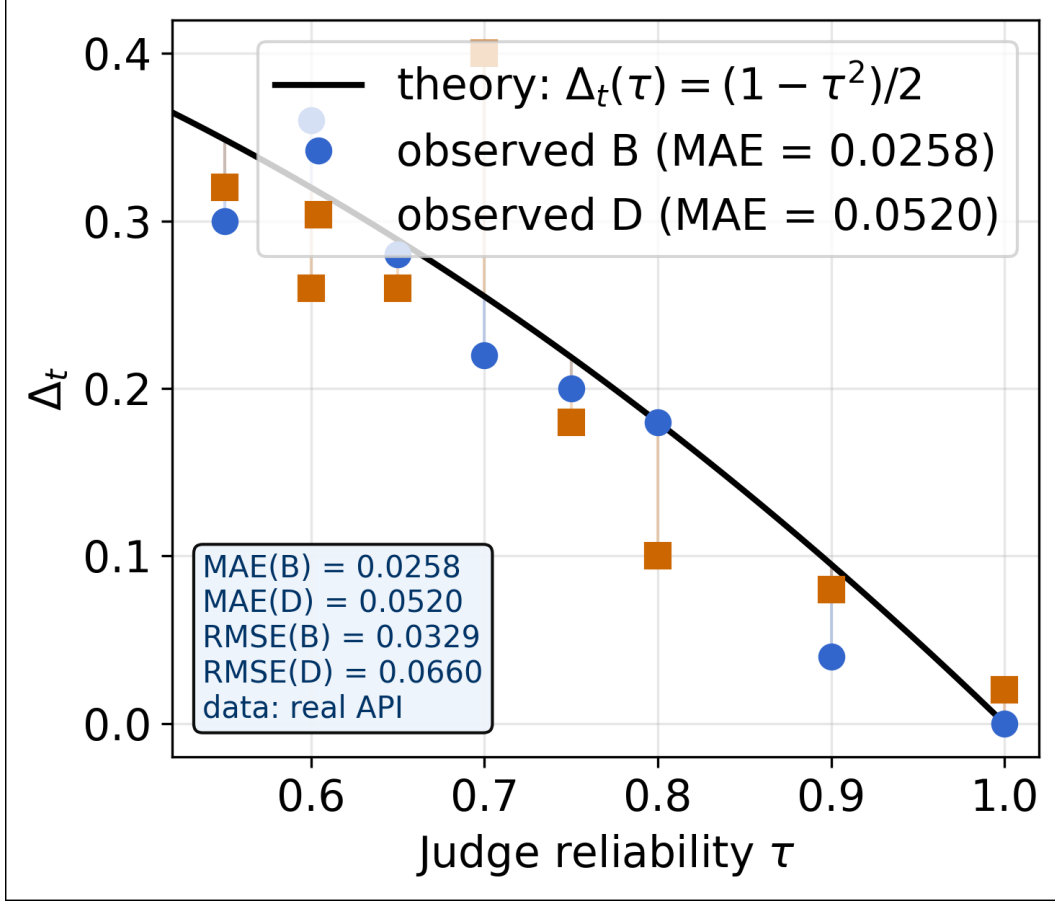


Figure 8: The closed form $\Delta_t(\tau) = (1 - \tau^2)/2$ against the observed scatter. The formula has no free parameters; measured MAE over 50 rounds per τ cell is 0.0258 (B) and 0.0520 (D). The curve is anchored at 0.5 rather than 1.0 when $\tau = 0$, which is the binary symmetry of two independent verdicts.

Over the 16 (τ , condition) cells, the two-judgment model is closer in 13, tied in 2 and loses in 1. The rejection is therefore not a preference between two equally good fits: the acceptance model is worse by a factor of six in MAE.

Why the factor of two appears. The acceptance model has no *both-wrong* agreement path. In Section 5.3, a round is scored correct whenever the two verdicts **agree**, so the probability mass splits into three exhaustive paths: both unflipped, exactly one flipped, and both flipped. The first and the third agree with each other and contribute zero error; only the middle path, of probability $2p(1 - p) = (1 - \tau^2)/2$, carries bias. The acceptance model instead charges the both-wrong case as an error, and that is what doubles Δ_t . The exponent of τ is fixed by the number of independent judgments the system requires to agree — a property of the architecture, not of the judge.

At the grid's lower edge, $\tau = 0.55$, the rejected model predicts $\Delta_t = 1 - 0.55^2 = 0.6975$, against a measured 0.3000 (B) and 0.3200 (D) and a closed-form prediction of $(1 - 0.55^2)/2 = 0.3488$. At the boundary $\tau = 0.5$ the rejected model gives $\Delta_t = 0.75$ against 0.375 from the closed form — a clean factor of two, though the swept grid begins at $\tau = 0.55$ and therefore holds no measurement at that point. **No result in this paper uses $\Delta_t = 1 - \tau^2$.**

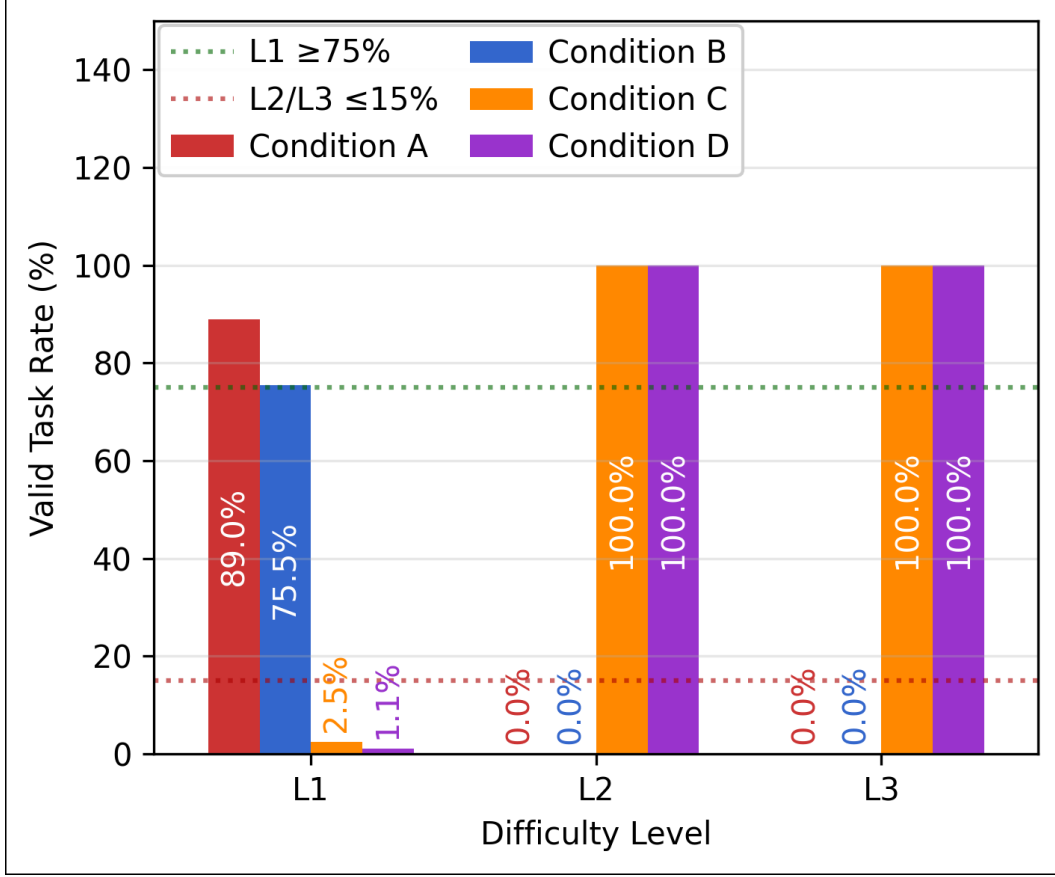


Figure 9: Valid-task rate by difficulty level. Every round that reached L2/L3 produced a valid task: the 85.5% and 90.5% invalid rates of C and D occur *before* level assignment, because the teacher fails at the L1→L2 transition while the controller holds the level high. This is the pseudo-difficulty trap.

D Data Consistency Check

D.1 Unified definition of Δ_t

$$\Delta_t = |\text{score}(y) - \text{label}(y^*)|$$

- y : the round’s submitted answer (`z3_result.status`);
- $\text{score}(y)$: the system’s normalised score for that answer, in $[0, 1]$;
- $\text{label}(y^*)$: the ground-truth label (`sat = 1.0`, `unsat = 0.0`).

D.2 Source of the system score by condition

Table D.1 records, for each condition, which quantity plays the role of the system’s evaluation and which plays the role of ground truth.

D.3 Two statistical calibers

Caliber 1: valid-only Δ_t (Table D.4). The denominator is `len(records[r.valid])`. Invalid rounds — those in which teacher generation failed and no ground-truth label exists — are excluded. Under this caliber, A gives $\Delta_t \approx 0.15$, since the LLM self-score deviates from the Z3 truth on a small number of rounds, while B and D give $\Delta_t \equiv 0.00$, because the Z3 verdict is compared against itself.

Table D.1: Where the system score and the ground-truth label come from in each condition. In B and D the Z3 verdict is compared against itself, so Δ_t is identically zero on every valid round; the global caliber of Section D.3 is what reintroduces a non-zero average.

Condition	System score $\text{score}(y)$	Ground truth label (y^*)	Δ_t
A	<code>llm_self_score</code> (LLM self-score)	expected (Z3 truth)	$ \text{llm_self_score} - \text{truth} $
B	Z3 anchor = <code>z3_result.status</code>	expected	0.00 — the Z3 verdict adopts the truth directly, so it is compared with itself
C	<code>llm_self_score</code> , averaged over the population	expected	$ \text{llm_self_score} - \text{truth} $
D	Z3 anchor + round-table consensus	expected	0.00 — the Z3 verdict is self-consistent

Table D.2: The same rounds under both calibers. The valid-only rows are the entries of Table D.4; the global rows additionally charge every invalid round $\Delta_t = 1.0$.

Caliber	Condition	n	Avg Δ_t
valid-only	A	534	0.1500
global	A	600	0.2435
valid-only	B	453	0.0000
global	B	600	0.2450
valid-only	C	87	0.1879
global	C	600	0.8823
valid-only	D	57	0.0000
global	D	600	0.9050

Caliber 2: global Δ_t (this appendix and Figure 4). The denominator is all rounds. Invalid rounds are encoded conservatively as $\Delta_t = 1.0$ and valid rounds keep their valid-only value. Under this caliber, A rises to ≈ 0.24 , moved by the 11% invalid rounds, and B rises to ≈ 0.25 , moved by the 24% invalid rounds.

D.4 Resolution of the apparent contradiction

A reader comparing the two reports will find $A = 0.15$, $B = 0.00$ in one and $A = 0.2435$, $B = 0.2450$ in the other. These are **not** in conflict: they are the same rounds read under the two calibers above, and Table D.2 verifies both simultaneously.

The paper uses **caliber 1 (valid-only)** for its main tables, and **caliber 2 (global)** when describing the pseudo-difficulty trap, where the quantity of interest is precisely how many rounds C and D invalidate.

Tables moved from the main text

Three tables that previously sat in the main text are collected here, so that the body keeps only the two tables it reasons about directly.

Table D.3: The composition law holds to machine precision on all four conditions (2,400 real-API rounds). Maximum absolute error 1.11×10^{-16} .

Condition	n	Δ_t^{gen}	Δ_t^{valid}	predicted Δ_t^{global}	observed	abs. error
A	600	0.1100	0.1500	0.2435	0.2435	5.55×10^{-17}
B	600	0.2450	0.0000	0.2450	0.2450	0
C	600	0.8550	0.1879	0.8822	0.8823	1.11×10^{-16}
D	600	0.9050	0.0000	0.9050	0.9050	0

Table D.4: Average Δ_t (valid-only caliber).

Condition	n (valid)	Avg Δ_t (valid only)	Std dev	Min	Max
A	534	0.1500	0.0000	0.1500	0.1500
B	453	0.0000	0.0000	0.0000	0.0000
C	87	0.1879	0.1394	0.1500	0.7000
D	57	0.0000	0.0000	0.0000	0.0000

Table D.5: Budget decomposition over the full sweep of Δ_t^{max} : the judge budget left over once the generator bottleneck is accounted for, and the judge reliability it requires. The main text states the $\Delta_t^{\text{max}} = 0.60$ case in prose; this is the complete sweep.

Condition	Δ_t^{gen}	Δ_t^{max}	judge budget	required τ_c
A	0.1100	0.60	0.5506	≥ 0 (random suffices)
B	0.2450	0.60	0.4702	0.244
C	0.8550	0.60	-1.7586	unreachable
D	0.9050	0.60	-3.2105	unreachable
A	0.1100	0.70	0.6629	≥ 0 (random suffices)
B	0.2450	0.70	0.6026	≥ 0 (random suffices)
C	0.8550	0.70	-1.0690	unreachable
D	0.9050	0.70	-2.1579	unreachable
A	0.1100	0.80	0.7753	≥ 0 (random suffices)
B	0.2450	0.80	0.7351	≥ 0 (random suffices)
C	0.8550	0.80	-0.3793	unreachable
D	0.9050	0.80	-1.1053	unreachable

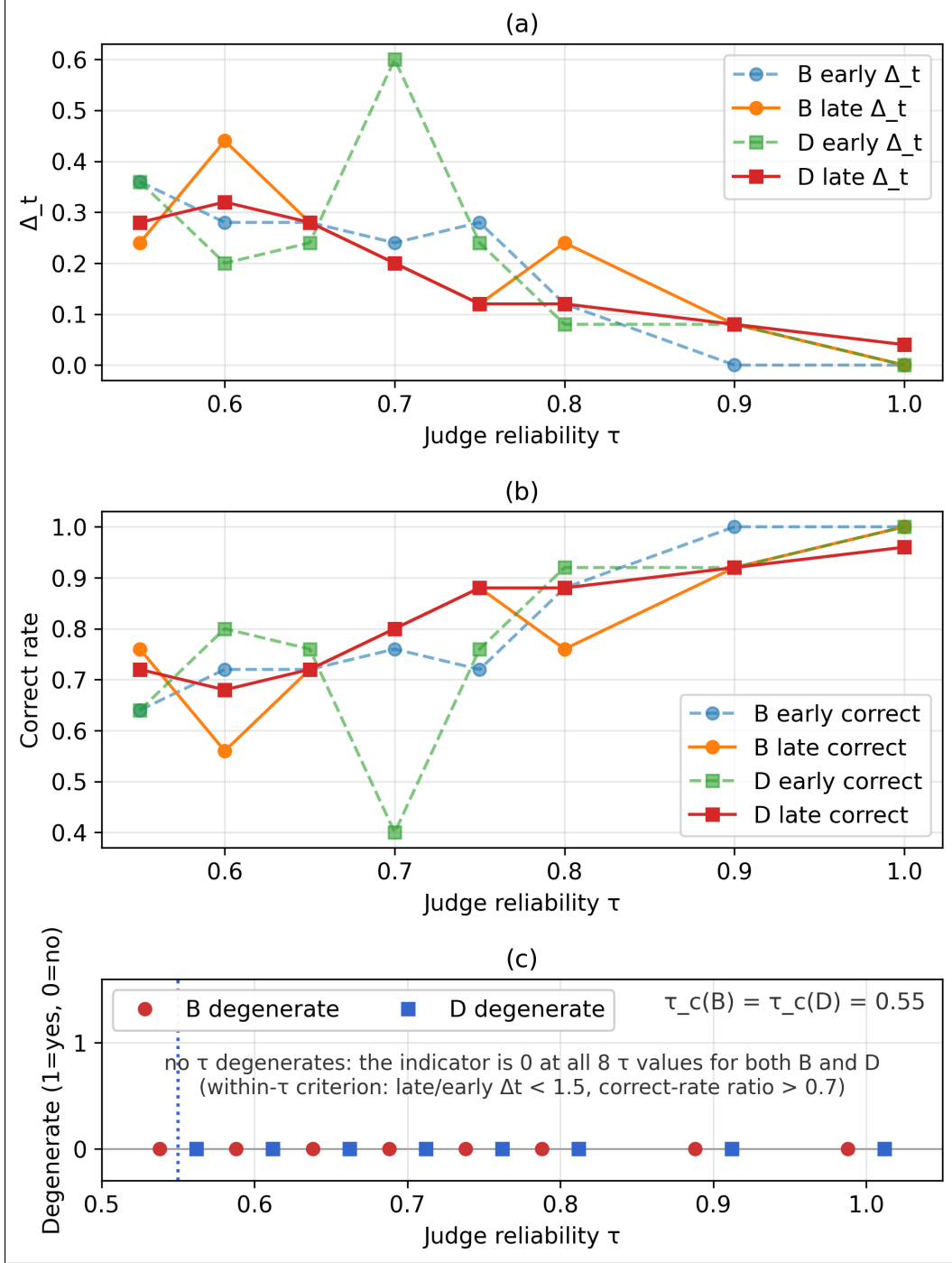


Figure 10: The τ -gradient sweep on the real API: 8 τ values $\times$ 2 conditions (B/D) $\times$ 50 rounds = 800 rounds. Under the within- τ degeneration criterion no τ value degenerates, so the reported $\tau_c = 0.55$ is the lower edge of the swept grid and should be read as an upper bound, not as the system's tolerance. Panel (c) plots the degeneration indicator: it is 0 at all eight τ values for both conditions, and the two vertical lines mark $\tau_c(B) = \tau_c(D) = 0.55$.

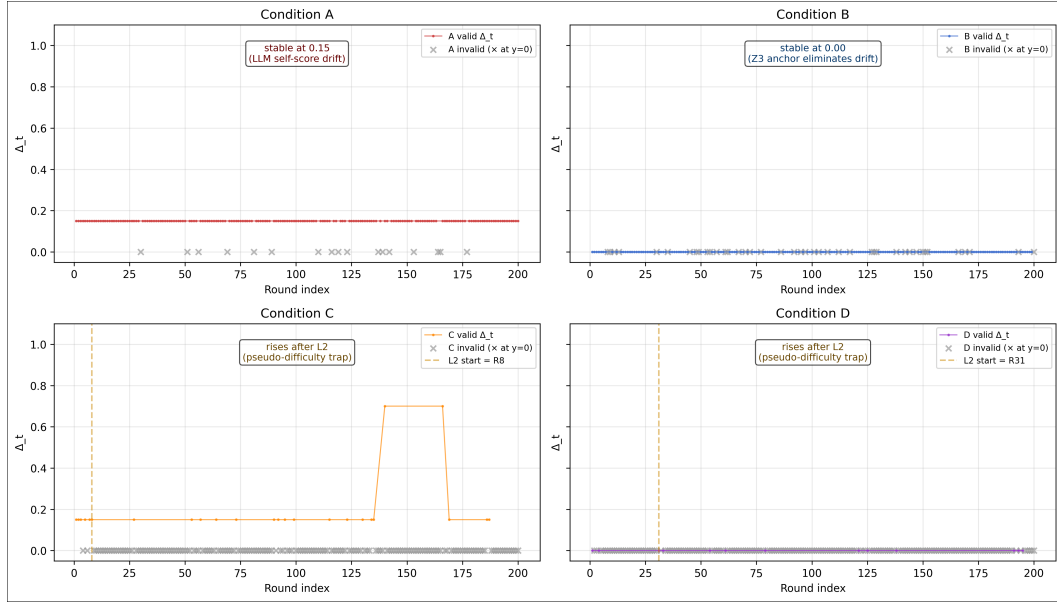


Figure 11: Each panel shows one condition; invalid rounds are marked with gray crosses. C/D show the pseudo-difficulty trap after L2 transition. Vertical lines mark the first round at which the difficulty controller raises the task level.