PERSPECTIVE

The Execution Layer Sets Agent Reliability and Cost, Not the Capability Ceiling: A Mechanistic Analysis of Interfaces, Verification and Attention Budgets

HX Research

Long-horizon agents now succeed or fail largely through an execution layer that sits outside the model: tool interfaces, verifiers, the scheduling of human attention and the metering of tool and data access. Practitioners routinely read gains from this layer as capability gains, treat tool calls as free and equate more human approval with more safety, yet no common metric links these choices. Here we argue that the execution layer sets an agent's reliability and unit cost rather than its capability ceiling. We organize the layer into four planes—representation, verification, scheduling and metering—and connect them through six equations covering compound per-step reliability, harness ceiling and mismatch, elicitation versus expansion, verifier precision, cost per verified success and oversight capacity. Synthesizing recent papers, official documentation and counter-evidence, we find that execution-layer gains mostly raise reliability and lower cost within the set of tasks a model can already reach, and that their direction depends strongly on the model. We give a diagnostic algorithm, a reproducible illustrative calculation and five falsifiable judgements, and propose managing the execution layer as controlled configuration budgeted in verified successes and human attention.

2 November 2026 • HX Research

Last weekend's news items look unrelated at first. The DeepSeek Harness team published a post proposing a Super-agent architecture. It argues that single tasks are increasingly bounded by model capability, whereas multitasking is bounded by human attention: a person can usually follow only three to five active sessions at once[1]. The team also open-sourced a research prototype[2]. At the same time, DeepSeek's API documentation stated that server-side search and fetch calls triggered when its API is used through DeepSeek Harness, Claude Code or Codex would be billed per successful request from 00:00 Beijing time on November 1[3]. On the Hugging Face trending list, one paper compared MCP and command-line tool interfaces under fully matched capabilities and found systematic differences in agent behaviour and cost[4]; another used an off-the-shelf frontier model, with no additional training, to solve nearly all of PutnamBench through recursive sub-agents and asynchronous search[5].

These events point to one question. Once the model is strong enough, which variables still decide whether an agent can deliver, how does each of them act, and where does each break down? We call these variables the execution layer: the software outside the model weights that determines how a model's output becomes a system action and how that action is confirmed—how tool interfaces present capabilities, how verifiers decide success, how schedulers allocate human and machine attention, and how metering prices each step. From ReAct, which interleaves reasoning and acting[6], and Toolformer, which lets a model teach itself to call APIs[7], to Gorilla and ToolLLM for large sets of real APIs[8,9] and the multi-agent conversation framework AutoGen[10], "model + execution layer" has become the standard form of an agent; benchmarks such as AgentBench, WebArena and SWE-bench likewise evaluate models inside complete execution environments[11–13].

Current discussion has three gaps. First, execution-layer gains are usually reported as single-attempt success (pass@1), which cannot tell whether a change makes reachable tasks more reliable or makes unreachable tasks reachable; recent work shows the two can reverse at larger attempt budgets[14]. Second, interfaces, verification, scheduling and metering are studied separately, without a framework that puts reliability, money and human attention in a single metric. Third, counter-evidence is under-cited: one study comparing MCP and CLI found that the agent

scaffolding mattered far more than the interface[15]; another, framed as a "bitter lesson", showed that a simple loop with an efficient verification interface can match elaborate workflows[16]. Kapoor and colleagues have argued that agent evaluations that look only at accuracy and ignore cost reach mistaken conclusions about where gains come from and make systems needlessly complex and costly[17].

Our central claim is that on long-horizon tasks the execution layer determines an agent's reliability and unit cost, not its capability ceiling. Its gains can be large, but they arrive mainly as elicitation and are tightly coupled to the specific model. We make three contributions:

1. We propose a four-plane framework for the execution layer and use four sets of numbered equations to place per-step reliability, elicitation versus expansion, cost per verified success and human oversight capacity under a single notation.
2. We analyse the representation, verification, scheduling and metering planes one by one, comparing technical routes, placing supporting and opposing evidence side by side, and stating where each route applies.
3. We give an executable algorithm for diagnosing and ranking execution-layer interventions (Algorithm 1), a reproducible illustrative calculation, and five forward-looking judgements, each with a time horizon, evidence, a falsification condition and enterprise implications.

Problem formulation and notation

Compounding per-step reliability

Let a task require L critical steps completed in sequence, and let s_t be the probability that step t succeeds given that all previous steps were correct. If any failure is fatal and unrecoverable, end-to-end success is

$$P = \prod_t s_t \approx s^{L}, \quad \partial \ln P / \partial s = L / s \tag{1}$$

Equation (1) says that small changes in per-step reliability are amplified by the number of steps. With L = 30, raising s from 0.97 to 0.98 lifts P from 0.401 to 0.545, a relative gain of about 36%; lowering s from 0.95 to 0.94 drops P from 0.215 to 0.156. Equation (1) is conservative. Real agents recover from errors, in which case the effective s_t is the complement of "erred and was not corrected"; it also ignores correlation between steps. A reliability study found that capability and reliability rankings diverge with multi-rank inversions as task duration grows, and that frontier models, which more often attempt ambitious multi-step strategies, show meltdown rates of up to 19% in some domains (according to the paper)[18]. Measurements of long tasks support this from another angle: according to the paper, the length of tasks frontier models complete with 50% success has doubled roughly every seven months since 2019, which the authors attribute mainly to better reliability and error recovery[19]; τ-bench measures consistency over repeated trials with pass^k and, according to the paper, gpt-4o has pass^8 below 25% in the retail domain[20].

Harness ceiling and policy mismatch

Denote the execution layer (the "harness" in the paper) by H and the model policy by π. Under KL-regularized control, Mu and colleagues show that deployed return decomposes as[21]

$$J(\pi, H) = U(H) - \Delta(\pi, H) \tag{2}$$

where U(H) is the best return the harness offers—its ceiling—and Δ(π, H) is the return lost to the policy's own suboptimal decisions—its mismatch. Training changes only Δ; changing the harness moves both U and Δ. The decomposition explains a frequently misread result: comparing two harnesses with a fixed model measures ceiling minus mismatch, not the quality of the harness itself.

Elicitation and expansion

Let p_i be the single-attempt success rate on task i. With a reliable verifier and k independent attempts,

$$pass@k = E_i[1 - (1 - p_i)^{k}], \quad \lim_{k\to\infty} pass@k = Pr(p_i > 0) \tag{3}$$

The limit $Pr(p_i > 0)$ is reachable-task coverage. Two kinds of improvement follow. Raising p_i only on tasks that are already reachable is elicitation: its benefit is large at small k and vanishes at large k. Raising coverage is expansion: it still pays at large k. Kim and colleagues applied this split to harness optimization and found that the pass@1 advantage of harnesses over stronger interventions, such as scaling the model or adding inference compute, can reverse at larger attempt budgets, and that aggregate gains can conceal shortfalls on hard tasks[14]. Eq. (3) uses the unbiased pass@k estimator of Chen and colleagues[22]. Studies of repeated sampling show that coverage grows roughly log-linearly with the number of samples over four orders of magnitude, translating directly into success in automatically verifiable domains such as code and formal proof[23]. Yue and colleagues use pass@k at large k to test reinforcement learning with verifiable rewards and find that it does not induce reasoning patterns beyond the base model: the trained model leads at small k and the base model overtakes it at large k (according to the paper)[24]. This is mechanistically the same as our reading of execution-layer gains as elicitation.

When the verifier is imperfect, let α be its acceptance rate on correct outputs and β its false-acceptance rate on incorrect ones. The share of accepted outputs that are actually correct is

$$Prec = \alpha P / [\alpha P + \beta(1 - P)] \quad (4)$$

With P = 0.40, α = 0.9 and β = 0.1, Prec ≈ 0.857; with the same verifier and P = 0.55, Prec ≈ 0.917. Verifier quality and base success are complements: the lower the base success, the more errors the same verifier lets through. Training verifiers to rank candidate solutions and supervising each step of the process are the two classic routes to raising α and lowering β[25,26]. An LLM judge is also a verifier: its agreement with human preferences can exceed 80%, but it shows position, verbosity and self-preference biases[27], which are exactly sources of β.

Cost per verified success

Let c_{tok} be the token cost of one attempt, with n_s search calls and n_f fetch calls priced at p_s and p_f, and let each failed attempt incur a human review cost h. If failed attempts can be retried until one is verified, the number of attempts is geometric with mean 1/P, so

$$C* = (c_{tok} + n_s \cdot p_s + n_f \cdot p_f) / P + h \cdot (1 - P) / P \quad (5)$$

Equation (5) puts metering and reliability in one expression. Cutting per-attempt cost without changing P lowers C∗ linearly; raising P lowers machine and human cost at once, and the human term falls faster.

Human oversight capacity

Let a reviewer have A minutes of effective attention per day, spend τ minutes per escalation on average, and face e expected escalations per task. The task throughput one reviewer can support is bounded by

$$N_{max} = A / (\tau \cdot e), \quad e = (1 - P) + r \quad (6)$$

where r is the spot-check rate on successful tasks. Equation (6) assumes review quality does not change with load. Turan shows that this assumption fails: when reviewers fatigue as escalation load grows, realized safety is an inverted U in the escalation rate, and the safety-optimal rate lies below full escalation[28]. Table 1 summarizes the notation. Classic human-factors research showed long ago that as automation increases, people swing between misuse, disuse and abuse[29]; and the more reliable the automation, the more easily supervisors lose the skill and vigilance needed to take over in an anomaly—the "ironies of automation"[30].

Table 1 | Main notation

Symbol	Meaning	First used
L, s_t, P	Critical steps, per-step success, end-to-end success	Eq. (1)
H, π, U, Δ	Harness, policy, harness ceiling, policy mismatch	Eq. (2)
p_i, k, pass@k	Single-attempt success on task i, attempts, share of tasks with at least one success in k	Eq. (3)
α, β, Prec	Verifier true-acceptance rate, false-acceptance rate, share of accepted outputs that are correct	Eq. (4)
c_{tok}, n_s, n_f, p_s, p_f, h, C∗	Token cost, search and fetch counts and prices, human review cost, cost per verified success	Eq. (5)
A, τ, e, r, N_{max}	Daily review attention, time per escalation, escalations per task, spot-check rate, throughput bound	Eq. (6)

Analytical framework: four planes of the execution layer

We split the execution layer into four planes that are distinct yet coupled. Each plane acts on different variables in Eqs. (1)–(6) and fails in different ways.

The representation plane determines what capabilities look like to the model: tool descriptions, argument schemas, return formats, the order of observation fields. It acts mainly on per-step success s_t and on policy mismatch Δ. The verification plane determines how success is judged: compilers, proof checkers, unit tests, state queries, independent review. It acts on α, β and the attempt budget k. The scheduling plane determines how tasks, sessions and people are organized: one session or a router–worker split, event-driven or instruction-driven, and how an external watchdog detects states that have not crashed but are making no progress. It acts on e, τ and A. The metering plane determines what each step costs and whether access is allowed at all: per-call tool billing, budget awareness, consent signals for data access. It acts on c_{tok}, p_s and p_f, and on the future availability of training data.

Table 2 | Four planes of the execution layer: variables, representative evidence and typical failures

Plane	Main variables	Representative evidence (past 12 months)	Typical failure
Representation	s_t, Δ	Matched MCP–CLI comparison; reordered observation fields; defective tool descriptions	Trained policies overfit to a representation; descriptions disagree with behaviour
Verification	α, β, k	Lean proving systems; independent review; agentic verification	Verifier bypassed; consensus hides errors; verification cost runs away
Scheduling	e, τ, A	Super-agent two-level architecture and watchdog; inverted-U oversight capacity	Mixed contexts; "alive but not progressing"; reviewer fatigue
Metering	c_{tok}, p_s, p_f	Per-request search/fetch billing; budget-aware tool use; resource-amplification attacks; training-preference signals	Budget blindness; malicious tools inflate cost; shrinking data availability

The planes interact. Better representation often lowers metered cost by cutting wasted calls; stronger verification raises metered cost but lowers C∗ by raising P; scheduling decides who handles a failed verification and how quickly. The sections below take each plane in turn, in the order mechanism, supporting evidence, opposing evidence and boundary of applicability.

Mechanisms and comparison of routes

Representation: same capabilities, different behaviour

The core question for the representation plane is whether agent behaviour changes when executable capabilities are identical and only their presentation to the model differs. Shi and colleagues built MICA, which renders MCP and CLI interfaces from one semantic action contract and routes decoded actions through the same runtime[4]. According to the paper, across six models and three interactive environments the two interfaces produced clear differences in task performance and interaction cost; 94.3% of 2,004 paired natural trajectories showed structured divergence, and early differences propagated through later interaction. The authors therefore locate "interface realization" as a direct intervention boundary. Their DART method repairs interface-specific realization while preserving validated semantic commitments and, according to the paper, raised benchmark scores in all six settings by up to 9.0 points while lowering MCP token cost in all three environments. This direction was first systematized by SWE-agent, which treats agents as a new kind of "user", designs a dedicated agent–computer interface for them, and shows that interface design alone substantially changes performance on software-engineering tasks[31]; CodeAct unifies the action space as executable Python code and, according to the paper, outperforms JSON or text actions across 17 models[32].

Mechanistically this is not surprising. Equation (2) implies that a post-trained policy carries the preferences of the harness it was trained under. Mu and colleagues ran a clean counterfactual on MiniWoB++: reversing the order of observation fields leaves every optimal decision unchanged, yet a trained model's success falls from 85.0% to 19.2% (according to the paper)[21]. In other words, representation matters because the policy itself is overfitted to the representation. Tool descriptions are another weak point. An empirical study of 856 tools on 103 MCP servers found that 97.1% of descriptions had at least one defect and 56% did not state their purpose clearly; augmenting descriptions raised median task success by 5.85 percentage points, but also increased execution steps by 67.46% and regressed performance in 16.67% of cases (according to the paper)[33].

The opposing evidence is equally strong. Alier Forment and colleagues ran one fixed software task across seven agent scaffoldings and five models, judging completion by inspecting repository state rather than trusting the agent's self-report[15]. According to the paper, the dominant effect was the scaffolding: two CLI-only scaffoldings completed every run and were 5.0 to 28 times cheaper than the five that support MCP; thirteen strictly paired MCP-to-CLI cost ratios ranged from 0.43 to 29, with no stable direction. Failures were equally frequent under both interfaces; what differed was the cost of failure—12.9% of MCP spending bought no completed work, against 2.2% for CLI. Agents also frequently ignored the interface they were assigned, so comparisons that do not verify actual behaviour measure an unknown mixture. A component-level study of coding harnesses by Fan and colleagues draws a similar boundary: predefined tools help models with weak bash proficiency, whereas bash-capable models work effectively through a bash-only interface at substantially lower cost[34].

Taken together, the two bodies of evidence yield a more useful conclusion than "MCP versus CLI". Interface effects are real, but their direction depends on the combination of model, scaffolding and task. The stable regularity is whether the interface exposes the state and semantics an action needs. Wang calls this the gap between callability and operability: a tool call can be perfectly valid yet leave the agent unable

to decide what to do next—for example, when an external effect commits but its response is lost, committed and uncommitted states become indistinguishable to the agent[35]. Fault-injection work makes the same point. Under faults calibrated to the observed incident distribution of the open-source MCP ecosystem, task-completion scores fell from 8.08 to 3.23 out of 10; the 120B model retained no larger share of its fault-free score than the 20B model, while agent stacks differed markedly in how well they absorbed faults (according to the paper)[36]. This directly contradicts the intuition that larger models are more fault-tolerant.

Verification: the verifier decides how many attempts you can spend

The verification plane works through Eqs. (3) and (4): only with a reliable verifier do k attempts turn into pass@k, and the false-acceptance rate β sets how many errors get through. Formal proof is where verifiers are most reliable, and so where the role of the execution layer is easiest to see. EasyProver consists of recursive agents with isolated contexts, asynchronous search, parallel tactic probing and an elastic elaboration service. According to the paper, with an off-the-shelf frontier model and no additional training it proves 669 of 672 PutnamBench problems and all six IMO 2026 formalizations, and resolves 597 of 600 NTP4VC verification conditions by proof or refutation[5]. Because these benchmarks are near saturation, the authors released the harder Aegis, which converts competitive-programming problems into verification tasks over real Rust implementations and Lean specifications; on a stratified sample of 2,000 statements, with a 30-minute budget each, the system proves 1,054, or 52.7% (according to the paper). Executable proof assistants such as Lean 4 give the verification plane a checker with a near-zero false-acceptance rate[37], and the early GPT-f already closed the loop between language-model generation and formal verification[38].

This result is often read as "clever orchestration beats training". Two other studies support a more restrained reading. Moalla and colleagues' h1 gives the model only an efficient Lean 4 interface and keeps it working through verifier feedback and self-reflection, imposing no decomposition or search structure. According to the paper, at matched models it matches or outperforms specialized workflows such as Goedel-Architect and AlphaProof Nexus, and ablations show that "simplicity alone is not enough: an efficient Lean interface largely determines cost and solve rate"[16]. Ortega and colleagues' mini-Lean-agent gives the model a single bash tool, with a separate trusted evaluator checking the final artifact; according to the paper, it solves 7 of 12 Putnam 2025 problems under a 2-dollar-per-problem limit, and all 12 at an average of 1.28 dollars per problem with a different base model[39]. Together they suggest that where verifiers are reliable, what matters is the efficiency of verification feedback—the boundary between representation and verification—rather than the complexity of orchestration.

Verification also costs money. Rögnvaldsson and colleagues add a control plane that observes failed Lean attempts, estimates the success probability and cost of another attempt, and decides whether to continue or restart from a new decomposition; according to the paper, it cuts cost by 28.9% on average over a fixed-step baseline on a PutnamBench subset while preserving performance[40]. Where no formal verifier exists, verification must be designed. Zhang and colleagues compared 17 designs across four layers—initial instructions, mid-turn reminders, completion gates and independent review. According to the paper, independent review by a critic offered the better accuracy–cost trade-off, raising average accuracy on Terminal-Bench by 5.9 to 9.5 percentage points at 1.13 to 1.25 times baseline cost; completion gates could yield larger gains but might raise cost substantially[41]. VeriHarness points to where β in Eq. (4) comes from: across repeated samples, disagreement often exposes correct alternatives, while consensus can conceal errors[42].

The boundary of the verification plane is that width buys rate, not budget. Bobadilla-Suarez and colleagues note that best-of-k orchestration realizes exactly the ceiling of the best worker; on 30 same-family workers failure overlap sits at its maximum, and a majority vote fails 23 of 55 tasks (42%, according to the paper)[43]. This agrees with Eq. (3): p_i for same-family models are highly correlated, so raising k mostly lifts success on reachable tasks without extending coverage. Research on test-time compute gives the same boundary: according to the paper, allocating inference compute adaptively by problem difficulty can be more cost-effective than scaling parameters, but the gain depends on the base model's existing success on the problem and on the verifier available[44]; methods such as Reflexion that self-correct through verbal feedback likewise presuppose an external feedback signal[45].

Scheduling: the bottleneck moves from the model to human attention

The scheduling plane asks who watches a portfolio of tasks once single tasks need less and less watching. DeepSeek's Super-agent post gives an engineering answer[1]. It decouples the agent from any single manually driven session and proposes four

characteristics (continuous, productive, proactive, persistent) and five modules: a two-level architecture in which a long-lived Router layer handles all human interaction and global coordination but executes no tasks itself, while a context-isolated Worker layer executes with high concurrency; event-driven design that treats environmental events—group-chat messages, emails, calendar reminders, code-review comments, CI results—as Triggers; multi-channel full-duplex interaction; separation of compute, state and environments; and an independent Watchdog session running outside the main system. The post stresses that many anomalies are not crashes: the process is still running but stuck in an unproductive loop or drifting from its goal, so the watchdog must monitor task progress as well as infrastructure liveness, using the model to judge whether each Worker is making substantive progress. The post states explicitly that it discusses architectural ideas, not existing product features, and the open-source prototype is declared research code that differs from both the internal version and the implementation described in the post[2].

Seen through Eq. (6), the design does three things. The two-level split lets people interact with a single Router, lowering the handling time τ per escalation; event triggers and the watchdog reduce anomalies that a human must discover, lowering e; and a Router that executes nothing avoids becoming a single-point bottleneck. Srinivasan offers an isomorphic abstraction from distributed systems: the load-bearing primitive of production agent runtimes is the stochastic–deterministic boundary, a four-part contract among proposer, verifier, commit step and reject signal; six runtime patterns, including event-driven sequencing, supervisor plus gate and human in the loop, compose that boundary differently[46]. The paper also names a failure mode, replay divergence, in which LLM-based consumers of the same deterministic event log produce different downstream outputs—an explicit requirement for auditing and reproducing event-driven architectures.

Opposing evidence on scheduling comes from modelling the human. Turan asked reviewers to label the risk of 125 adversarially weighted agent actions and obtained a Fleiss κ of only 0.52, so "risky" has no single correct label; once reviewers are modelled as fatiguing with escalation load, realized safety becomes an inverted U in the escalation rate (according to the paper)[28]. "Have a person approve every action" is therefore not a safe default. Khanal and colleagues' reliability study also found that memory scaffolds hurt long-horizon performance across all ten models evaluated[18], a reminder that persistent cross-session context is not automatically beneficial and that a Super-agent's Router memory needs to be evaluated on its own.

Metering: tool calls and data access are acquiring prices

The metering plane has changed most concretely in recent days. DeepSeek's official documentation states that the Search and Fetch APIs are billed per successful request, not by tokens or by the number of returned results, at 0.75 US dollars per 1,000 successful requests each; only requests that return HTTP 200 and produce a result are billable, while failed requests and empty results are not; and server-side tool calls made when the DeepSeek API is used through DeepSeek Harness, Claude Code or Codex, previously free, are billed per request from 00:00 Beijing time on November 1, 2026 (according to the official documentation)[3,47]. The same pricing page lists deepseek-flash at 0.3 dollars per million input tokens (cache miss) and 1.2 dollars per million output tokens at peak, with off-peak rates at half (according to the official documentation)[47]. Media reports of a renminbi price have not been verified against the official documentation; we use only the dollar prices published there.

Billing changes not just the invoice but the agent's optimal strategy. Liu and colleagues found that simply raising the tool-call budget does not improve web-search agents, because they lack budget awareness and quickly hit a ceiling; giving them continuous budget tracking, and adapting planning and verification to it, pushes out the cost–performance Pareto frontier (according to the paper)[48]. Metering also opens a new attack surface. Zhou and colleagues built a malicious MCP server that edits only text-visible fields, keeping function signatures and the final benign payload, yet steers agents into prolonged calling chains; according to the paper, per-query cost rises by up to 658 times, and standard prompt filters and output monitors seldom notice[49]. Tool-call traces themselves leak user profiles: on 4,000 trajectories generated with real tools from 25 MCP servers, attackers inferred profiles with a 26.25% success rate, against 17.97% from prior-based guessing (according to the paper)[50]. This attack surface shares its origin with indirect prompt injection: data returned by tools can hijack an agent's subsequent behaviour[51]; with 97 tasks and 629 security test cases, AgentDojo shows that neither current attacks nor defences have settled the problem[52].

The metering plane extends upstream to training data. A longitudinal audit of 14,000 web domains by Longpre and colleagues showed that restrictions on data use rose rapidly between 2023 and 2024, fully restricting about 5% or more of tokens in C4, and 28%

or more of its most actively maintained critical sources (according to the paper)[53]. On October 30, OpenAI, Google, Microsoft and Anthropic said they would honour a shared "policy token" placed in robots.txt and based on a proposal in the IETF AIPREF working group; according to the report, Anthropic has already adopted the signal, OpenAI says it will implement it in the "coming months", Google and Microsoft plan to support it next year, and Meta signalled support without saying whether it will honour it[54]. Google's official blog explains that when a site uses the token to opt out of training and does not explicitly specify Google-Extended, Google plans to treat it as disallowing Google-Extended, and aims to support the control in 2027[55]. Academic proposals go further: terms.txt expresses per-path, per-purpose machine-access terms in a robots.txt-style file and negotiates price through signatures, delegation tokens and HTTP 402[56]. Per-call payment for live fetching plus preference-based consent for training crawls means that data availability is turning from a free default into a metered, consented execution-layer variable.

An echo on the training side: schedules replace staged algorithms

The execution-layer argument has a counterpart in training. Song and colleagues derive suitable reinforcement-learning objectives as maximum-likelihood fitting to model-generated target distributions, placing pretraining, midtraining, supervised fine-tuning and reinforcement learning under a single likelihood-based algorithm; the training lifecycle becomes a question of which data to train on, in what proportions and when, with staged training as a special case (according to the paper)[57]. Yu and colleagues find that the gains of weight-normalized training over weight decay come mainly from the effective-learning-rate trajectory it induces; matching that trajectory largely collapses the two loss curves. Their AIR schedule lowers terminal training loss by about 0.01 relative to cosine at matched cumulative learning rate in 128B-token MoE pretraining (according to the paper)[58]. Another study likewise observes loss trajectories collapsing when effective learning rates are matched[59].

The relation to our claim is that both works promote the schedule—of data mixture and of effective learning rate—to a first-class design object. They also mark the boundary of our claim. In Eq. (2) only training lowers mismatch Δ and only the harness raises ceiling U. Mu and colleagues show that a new harness rejected by a fixed-model comparison but offering a higher ceiling outperforms the original agent after further training under it[21]. The execution layer and training are therefore not substitutes but a division of labour—the harness sets the ceiling, training removes the mismatch—and optimizing them separately systematically underestimates their joint gain.

Table 3 | Comparison of execution-layer intervention routes (all evidence self-reported in papers or official documentation, not independently reproduced)

Route	Mechanism	Main benefit	Main cost or risk	Applies when	Does not apply when
Interface repair (representation)	Expose action-relevant state and semantics; fix description defects	Higher s_t, fewer wasted calls	Mismatch grows if it departs from the training representation	Action semantics can be contracted	Model works mainly through a generic shell
Simple loop + efficient verification interface	Verifier feedback drives persistent attempts	Close to specialized workflows at low orchestration cost	Depends on a reliable verifier	Formal or executable-test domains	No objective criterion
Elaborate orchestration (decomposition, search)	Explicit decomposition and parallel search	Higher accuracy for weaker models	Mostly a cost saver for strong models; gains absorbed by model upgrades	Tight budget, weaker model	Frontier model with efficient interface
Independent review	A separate session checks the output	Accuracy gain at modest extra cost	Same-family reviewers fail together	Output can be checked against evidence	Reviewer shares the generator's source and has no external evidence
Two-level scheduling + watchdog	Separate routing from execution; monitor progress	Lower e and τ, more parallelism	Replay divergence, memory contamination	Event-driven, many tasks	Single short task
Budget-aware metering	Expose remaining budget to the agent	Better cost–performance frontier	The budget policy itself needs evaluation	Tools billed per call	Tool cost negligible

Method: diagnosing and ranking execution-layer interventions

The analysis above reduces to an executable procedure. Algorithm 1 decides, for a given model and task set, whether an execution-layer intervention is elicitation or expansion, estimates its effect on cost per verified success C∗ and on human throughput

N_{max}, and ranks interventions in a statistically sound way.

Algorithm 1 Execution-Layer Diagnosis and Ranking (ELDR)

Input: Task set T, each task with an executable verifier V or a human judging rule; model M; candidate interventions {H_0, H_1, ..., H_m}, where H_0 is the current harness; attempt budget K; prices p_s, p_f and token prices; human review cost h; review attention A, handling time τ, spot-check rate r

Output: For each intervention: ΔP@1, ΔP@K, a type label (elicitation / expansion / none), C∗, N_{max}, and a ranking corrected for multiple testing

1: For each H_j and each task i, run K independent attempts; judge success with V; log token use, n_s, n_f, the failing step and failure type; judge completion from actual repository or system state, never from the agent's self-report

2: Estimate $\hat{p}_i$ = successes / K; compute pass@1 and pass@K by Eq. (3); compute coverage $\hat{C}$ = share($\hat{p}_i > 0$)

3: Classify: if ΔP@1 > 0 and the confidence interval of $\Delta\hat{C}$ contains 0, label "elicitation"; if $\Delta\hat{C} > 0$ is significant, label "expansion"; otherwise label "none"

4: Compute C∗ by Eq. (5) and N_{max} by Eq. (6); for interventions that include a verifier, estimate α and β on held-out samples and compute Prec by Eq. (4)

5: Run exact McNemar tests on paired task outcomes of H_j versus H_0; apply a multiple-testing correction across the m comparisons

6: Rank only interventions that pass step 5, by ascending C∗; flag those whose N_{max} falls below business demand as "needs scheduling support"

7: For each passing intervention, run one representation-perturbation test (for example reordering fields or rewriting descriptions) and record the performance drop as a risk label for "coupling to the current model"

Complexity and conditions: Requires $O(n_T \cdot (m+1) \cdot K)$ episodes (n_T is the number of tasks); K of 4 to 8 suffices to tell the direction of elicitation versus expansion, though coverage is underestimated at small K; requires reproducible verifiers and approximately independent tasks; not suitable for tasks with no objective criterion and low inter-rater agreement

Three design choices deserve comment. Step 1 judges completion from actual state because agents ignore their assigned interface or misreport completion[15], and self-reported success systematically inflates P. Step 5 uses paired tests with multiplicity correction because small leaderboard gaps are usually not enough to rank systems: an audit of SWE-bench found that exact McNemar tests at α = 0.05 separate none of the 29 adjacent pairs in the Verified top thirty, while the same model's score ranges by up to 29.8 percentage points across scaffolds (according to the paper)[60]. Step 7 follows directly from Eq. (2): if an intervention's gain largely disappears after reordering fields, it is reducing the current model's mismatch rather than raising the harness ceiling, and will probably need redoing when the model changes. Paired comparison uses McNemar's test[61], one of the procedures Dietterich recommends among approximate statistical tests for comparing learning algorithms[62].

Illustrative calculation

To show how Eqs. (1)–(6) interact, we use one set of illustrative data. Apart from DeepSeek's official unit prices, all parameters are assumptions; the script and random seed are in src/figs.py and the results are reproducible.

Reliability. Let a task have L = 30 critical steps. The baseline harness has per-step success s = 0.97, so $P = 0.97^{30} \approx 0.401$; an interface repair raises s to 0.98 and $P \approx 0.545$ (Fig. 1). Adding an independent review step, we assume P rises further to 0.62, a gain of the same order as that reported for independent review[41].

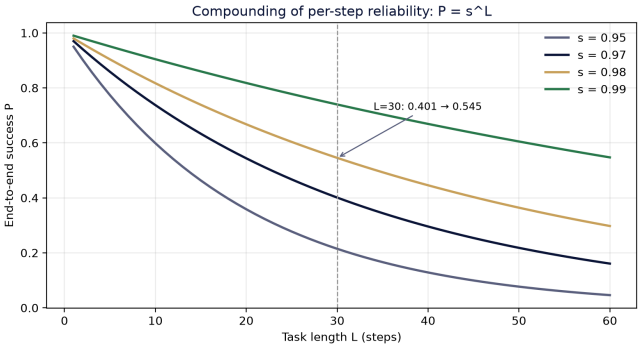


Fig. 1 | Compounding of per-step reliability. Curves show P = s^L from Eq. (1) for several values of s. At L = 30, raising s from 0.97 to 0.98 lifts P from 0.401 to 0.545. Illustrative data computed by the Great Unicorn Intelligence Research Team.

Cost. Assume each attempt uses 200,000 input tokens (all cache misses) and 20,000 output tokens; at the deepseek-flash peak price this is 0.2 × 0.3 + 0.02 × 1.2 = 0.084 dollars. Each attempt makes 20 search and 10 fetch calls; at 0.75 dollars per 1,000 this is 30 × 0.00075 = 0.0225 dollars (unit prices from the official documentation[47]). Under these assumptions tool calls are about 21% of per-attempt machine cost—not a rounding error. We assume a human review cost of h = 0.40 dollars per failed attempt and let independent review consume 20% more tokens. Results by Eq. (5) are shown in Table 4 and Fig. 2.

Table 4 | Illustrative calculation: cost per verified success under three execution-layer configurations (US dollars, illustrative data)

Configuration	P	Machine cost / P	Of which tool calls / P	Human term h(1−P)/P	C∗
Baseline	0.401	0.266	0.056	0.598	0.863
Interface repair	0.545	0.195	0.041	0.334	0.529
Interface repair + independent review	0.620	0.199	0.036	0.245	0.444

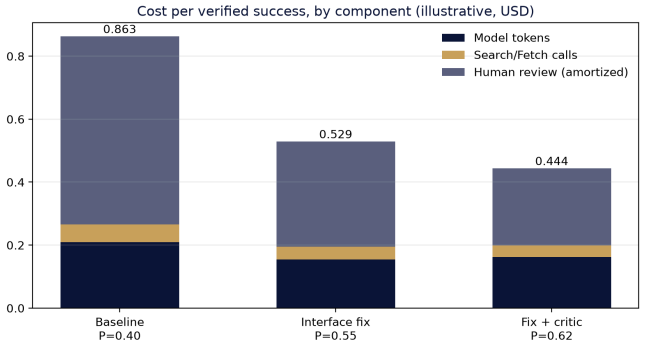


Fig. 2 | Composition of cost per verified success. At baseline, amortized human review is about 69% of C∗; interface repair lowers C∗ by about 39%, mostly through the human term. Illustrative data computed by the Great Unicorn Intelligence Research Team.

Table 4 contains two counter-intuitive points. First, independent review raises per-attempt machine cost, yet C∗ keeps falling, because by raising P it reduces both retries and human review. Second, when human review is priced well above an attempt's machine cost, C∗ is far more sensitive to P than to token prices: halving token prices lowers baseline C∗ only from 0.863 to 0.759, whereas raising P from 0.401 to 0.545 brings it to 0.529.

Oversight capacity. Let each reviewer have A = 360 minutes of effective review time per day, $\tau = 4$ minutes per escalation and a spot-check rate $r = 0.1$. By Eq. (6), at baseline $e = 0.599 + 0.1 \approx 0.70$ and $N_{max} \approx 360 / (4 \times 0.70) \approx 129$ tasks per day; with interface repair plus independent review, $e = 0.38 + 0.1 = 0.48$ and $N_{max} \approx 188$ tasks per day. This is only an upper bound under constant review quality; with fatigue, the escalation rate should be held near the peak of the inverted U rather than run at capacity[28].

Elicitation and expansion. To illustrate Eq. (3) we simulate 1,000 tasks: 70% are reachable, with single-attempt success drawn from Beta(2, 3), and 30% are unreachable. An "execution-layer optimization" raises success on reachable tasks by 60% (capped at 1) and leaves coverage unchanged; a "model upgrade" raises success on reachable tasks by only 15% but makes 15% of unreachable tasks reachable (single-attempt success 0.25). As Fig. 3 shows, the execution-layer optimization reaches pass@1 of 0.430, above the model upgrade's 0.335; at k = 32 the figures are 0.700 and 0.747, and the order reverses.

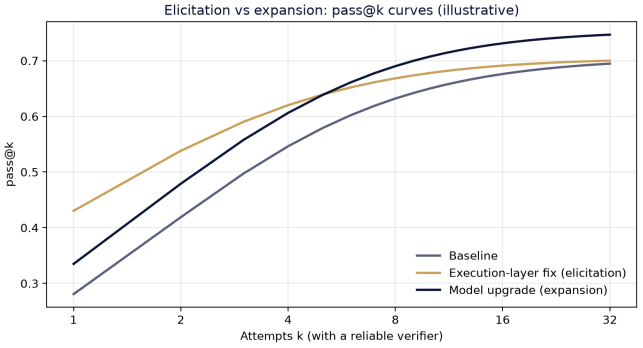


Fig. 3 | pass@k curves for elicitation and expansion. The execution-layer optimization leads at small k and the model upgrade overtakes it at large k, consistent with the phenomenon reported in the literature[14]. Illustrative data with a fixed random seed, computed by the Great Unicorn Intelligence Research Team.

The enterprise implication is that if a workflow allows multiple attempts and has a reliable verifier—reconciliation, rule checks, code tests—an improvement must be evaluated on both pass@1 and pass@k at larger k; if the workflow allows only one attempt, such as a real-time customer reply, elicitation-type improvements are worth more.

Forward-looking judgements and research agenda

The following judgements are the opinions of the Great Unicorn Intelligence Research Team, not statements of fact. Each gives a time horizon, evidence, a falsification condition and enterprise implications.

Table 5 | Five forward-looking judgements (opinion)

No.	Judgement	Horizon	Main evidence	Falsified if	Enterprise implication
1	Server-side tool calls on major agent platforms will generally be priced per call as a separate line, and enterprise evaluation will shift from "price per million tokens" to "cost per verified successful task"	6–12 months	Search and fetch billed per successful request[3]; budget-awareness research[48]	By end-2027, major providers still bundle server-side search for free without a separate billing line	Attribute token, tool and human cost per task; compare options by Eq. (5)
2	Representation items—tool descriptions, field order, return formats—will be brought under change management and regression testing like model versions	12–18 months	Matched interface comparison[4]; field reordering[21]; description defects[33]	Controlled studies covering five or more model families find representation perturbations generally move success by less than 2 points	Version interface definitions; any interface change triggers regression evaluation
3	In domains with programmatic verifiers, "simple loop + efficient verification interface" will become the default baseline, and elaborate orchestration will shrink to a cost-saving role	12–24 months	h1[16]; mini-Lean-agent[39]; component-level study[34]	At matched model and compute, elaborate decomposition workflows keep an advantage above 10 points	Invest in verifiers and feedback interfaces before orchestration
4	Human oversight will move from per-action approval to load-based escalation budgets, with oversight capacity an explicit design parameter	12–24 months	Inverted-U oversight capacity[28]; Super-agent attention constraint and watchdog[1]	Studies under realistic fatigue show full escalation is still safety-optimal	Set escalation quotas for review roles; plan staffing by Eq. (6)
5	The execution-layer assets that survive model generations will be mainly verifiers and state semantics, not prompts and workflow orchestration	24–36 months	Policies absorb harness preferences[21]; elicitation gains reverse at large k[14]; width buys rate only[43]	Prompt- and workflow-type optimizations retain more than 80% of their gains after a model generation change	Prioritize budget for verifiers, state queries and idempotent semantics

Around these judgements we see four research questions worth investment: standardized perturbation tests for execution layers that make "coupling to the current model" a reportable metric; verifiers with estimable false-acceptance rate β for business domains without formal verifiers, such as compliance review and report writing; incorporating reviewer fatigue into offline evaluation of oversight policies; and joint optimization of the execution layer and post-training, to test whether the division "the harness sets the ceiling, training removes the mismatch" holds at larger scale.

Where common industry views may be wrong

First, "MCP is better than CLI", or the reverse. Interface effects are real, but their direction varies with model, scaffolding and task, and strictly paired cost ratios range from 0.43 to 29[15]. The more reliable question is whether the interface exposes the state and semantics an action needs.

Second, "stronger models are more fault-tolerant". Under realistically distributed tool faults, the larger model retained no larger share of its fault-free score, and framework-level error handling mattered more[36].

Third, "more human approval is safer". When reviewers fatigue, safety is an inverted U in the escalation rate[28].

Fourth, "execution-layer optimization equals capability gain". Most execution-layer gains are elicitation, and a pass@1 advantage can reverse at larger attempt budgets[14].

Fifth, "small leaderboard gaps are a ranking". Most adjacent pairs in the top thirty cannot be separated statistically[60].

Sixth, "tokens are the only variable cost". Tool calls are already priced per call[3], and malicious tools can inflate per-query cost by hundreds of times[49].

Limitations

This analysis has four limitations. First, most cited studies are recent preprints whose numbers are self-reported and not yet peer-reviewed or independently reproduced; results from formal proving may not transfer to business tasks without reliable verifiers. Second, Eq. (1) assumes independent, unrecoverable steps, Eq. (5) assumes independent retries and Eq. (6) assumes constant review quality; all hold only approximately in real systems. Third, the token usage, call counts, review cost and success gains in the illustrative calculation are assumptions used to show mechanisms; they do not represent measured performance of any specific system. Fourth, the Super-agent post is an architecture essay with no public quantitative evaluation of its internal instance, so we discuss only its design logic.

Conclusion

To return to the central claim: on long-horizon tasks the execution layer determines an agent's reliability and unit cost, not its capability ceiling. The representation plane amplifies small differences through compounding per-step success, but the direction of its effect depends on model and scaffolding. The verification plane decides whether an attempt budget turns into success, and a reliable verifier with an efficient feedback interface often

beats elaborate orchestration. The scheduling plane moves the bottleneck from the model to human attention, so oversight capacity must be designed explicitly. The metering plane puts prices on tool calls and on data access. Unified training and effective-learning-rate studies suggest that the execution layer and training should be seen as a division of labour between setting the ceiling and removing the mismatch.

In Great Unicorn Intelligence's practice, these principles sit in the governance layer of the Enterprise Agent Platform Harness. According to the company website, the platform defines an agent as "brain + hands + reins" and organizes it into three layers and twelve components—cognition, knowledge and action, and governance. Controlled tool calls via MCP/API, a rules engine, permissions re-checked on every call and a replayable audit trail correspond to the representation, verification and scheduling planes discussed here; the model only proposes and the system decides[63].

Key takeaways:

1. Read reliability through P = s^L: on long tasks, small per-step improvements are amplified by the number of steps.
2. Separate elicitation from expansion: report pass@1 and pass@k at large k, and do not count harness gains as capability gains.
3. Budget in cost per verified success C∗, including human review and tool calls.
4. Treat human attention as a scarce resource: set load-based escalation budgets instead of per-action approval.
5. Invest first in verifiers and state semantics: they are more likely than prompts and workflows to keep their value across model generations.

References

1. DeepSeek Harness Team. Exploring Super-agent Design: Towards a More Autonomous Agent System. X (DeepSeek Harness), https://x.com/DeepSeekHarness/status/2116857257245987236 (2026).
2. DeepSeek. dsh-superagent-prototype: Towards a More Autonomous Agent System (research prototype). GitHub, https://github.com/deepseek-ai/dsh-superagent-prototype (2026).
3. DeepSeek. Search & Fetch. DeepSeek API Docs, https://api-docs.deepseek.com/guides/search/ (2026).
4. Shi, Z., Pang, L. & Li, Y. Matched Capabilities, Different Behaviors: A Controlled Study of MCP and CLI Interfaces for Language Agents. Preprint at https://arxiv.org/abs/2610.34298 (2026).
5. Ganguly, D. et al. Scaling Lean Theorem Proving for Real-World Code Verification. Preprint at https://arxiv.org/abs/2610.38174 (2026).
6. Yao, S. et al. ReAct: Synergizing Reasoning and Acting in Language Models. In *Proc. International Conference on Learning Representations* (2023); https://arxiv.org/abs/2210.03629.
7. Schick, T. et al. Toolformer: Language Models Can Teach Themselves to Use Tools. In *Advances in Neural Information Processing Systems* **36** (2023); https://arxiv.org/abs/2302.04761.
8. Patil, S. G., Zhang, T., Wang, X. & Gonzalez, J. E. Gorilla: Large Language Model Connected with Massive APIs. In *Advances in Neural Information Processing Systems* **37** (2024); https://arxiv.org/abs/2305.15334.
9. Qin, Y. et al. ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs. In *Proc. International Conference on Learning Representations* (2024); https://arxiv.org/abs/2307.16789.
10. Wu, Q. et al. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. Preprint at https://arxiv.org/abs/2308.08155 (2023).
11. Liu, X. et al. AgentBench: Evaluating LLMs as Agents. In *Proc. International Conference on Learning Representations* (2024); https://arxiv.org/abs/2308.03688.
12. Zhou, S. et al. WebArena: A Realistic Web Environment for Building Autonomous Agents. In *Proc. International Conference on Learning Representations* (2024); https://arxiv.org/abs/2307.13854.
13. Jimenez, C. E. et al. SWE-bench: Can Language Models Resolve Real-World GitHub Issues? In *Proc. International Conference on Learning Representations* (2024); https://arxiv.org/abs/2310.06770.
14. Kim, Y. et al. Elicitation or Expansion? Demystifying Gains from Agent Harness Optimization. Preprint at https://arxiv.org/abs/2610.27889 (2026).
15. Alier Forment, M., Casañ Guerrero, M. J., García-Peñalvo, F. J. & Pereira, J. The Scaffolding Matters More Than the Interface: A Controlled Comparison of MCP and CLI Tool Use Across Seven Agent Scaffoldings, Five Language Models, and One Software Task. Preprint at https://arxiv.org/abs/2608.08654 (2026).
16. Moalla, S., Rammal, A., Patel, N., Munos, R. & Kempe, J. h1: A Taste of the Bitter Lesson in Automated Theorem Proving. Preprint at https://arxiv.org/abs/2610.35962 (2026).
17. Kapoor, S., Stroebl, B., Siegel, Z. S., Nadgir, N. & Narayanan, A. AI Agents That Matter. Preprint at https://arxiv.org/abs/2407.01502 (2024).
18. Khanal, A., Tao, Y. & Zhou, J. Beyond pass@1: A Reliability Science Framework for Long-Horizon LLM Agents. Preprint at https://arxiv.org/abs/2603.29231 (2026).
19. Kwa, T. et al. Measuring AI Ability to Complete Long Software Tasks. In *Advances in Neural Information Processing Systems* **38** (2025); https://arxiv.org/abs/2503.14499.
20. Yao, S., Shinn, N., Razavi, P. & Narasimhan, K. τ-bench: A Benchmark for Tool-Agent-User Interaction in Real-World Domains. Preprint at https://arxiv.org/abs/2406.12045 (2024).
21. Mu, J., Shen, R. & Shu, Y. Your Policy Is Secretly a Harness: The Decision Geometry of Agent Interfaces. Preprint at https://arxiv.org/abs/2610.37255 (2026).
22. Chen, M. et al. Evaluating Large Language Models Trained on Code. Preprint at https://arxiv.org/abs/2107.03374 (2021).
23. Brown, B. et al. Large Language Monkeys: Scaling Inference Compute with Repeated Sampling. Preprint at https://arxiv.org/abs/2407.21787 (2024).
24. Yue, Y. et al. Does Reinforcement Learning Really Incentivize Reasoning Capacity in LLMs Beyond the Base Model? In *Advances in Neural Information Processing Systems* **38** (2025); https://arxiv.org/abs/2504.13837.
25. Cobbe, K. et al. Training Verifiers to Solve Math Word Problems. Preprint at https://arxiv.org/abs/2110.14168 (2021).
26. Lightman, H. et al. Let's Verify Step by Step. In *Proc. International Conference on Learning Representations* (2024); https://arxiv.org/abs/2305.20050.
27. Zheng, L. et al. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems* **36** (Datasets and Benchmarks Track) (2023); https://arxiv.org/abs/2306.05685.
28. Turan, E. Oversight Has a Capacity: Calibrating Agent Guards to a Subjective, Fatiguing Human. Preprint at https://arxiv.org/abs/2606.08919 (2026).
29. Parasuraman, R. & Riley, V. Humans and Automation: Use, Misuse, Disuse, Abuse. *Hum. Factors* **39**, 230–253 (1997); https://doi.org/10.1518/001872097778543886.
30. Bainbridge, L. Ironies of automation. *Automatica* **19**, 775–779 (1983); https://doi.org/10.1016/0005-1098(83)90046-8.

31. Yang, J. et al. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems* **37** (2024); https://arxiv.org/abs/2405.15793.
32. Wang, X. et al. Executable Code Actions Elicit Better LLM Agents. In *Proc. 41st International Conference on Machine Learning* (PMLR, 2024); https://arxiv.org/abs/2402.01030.
33. Hasan, M. M., Li, H., Rajbahadur, G. K., Adams, B. & Hassan, A. E. Model Context Protocol (MCP) Tool Descriptions Are Smelly! Towards Improving AI Agent Efficiency with Augmented MCP Tool Descriptions. Preprint at https://arxiv.org/abs/2602.14878 (2026).
34. Fan, R. et al. An Empirical Study of Harness Design for Coding Agents. Preprint at https://arxiv.org/abs/2609.20804 (2026).
35. Wang, Z. Callability Is Not Operability: Controlled Interface Interventions for LLM Agents. Preprint at https://arxiv.org/abs/2608.23628 (2026).
36. Iannillo, A. K. et al. Tool-Calling LLM Agents Under Realistic MCP Faults: Scale Does Not Purchase Fault Tolerance. Preprint at https://arxiv.org/abs/2610.32555 (2026).
37. de Moura, L. & Ullrich, S. The Lean 4 Theorem Prover and Programming Language. In *Automated Deduction – CADE 28*, Lecture Notes in Computer Science **12699**, 625–635 (Springer, 2021); https://doi.org/10.1007/978-3-030-79876-5_37.
38. Polu, S. & Sutskever, I. Generative Language Modeling for Automated Theorem Proving. Preprint at https://arxiv.org/abs/2009.03393 (2020).
39. Ortega, T. et al. Do Theorem Provers Need Theorem-Proving Agents? mini-Lean-agent: A Coding-Agent Baseline for Lean. Preprint at https://arxiv.org/abs/2610.17331 (2026).
40. Rögnvaldsson, K., Sun, C., Dekoninck, J. & Vechev, M. Optimizing the Cost-Quality Tradeoff of Agentic Theorem Provers in Lean. Preprint at https://arxiv.org/abs/2606.04883 (2026).
41. Zhang, Z. et al. Rethinking Verification in Agent Harnesses. Preprint at https://arxiv.org/abs/2610.34577 (2026).
42. Zhang, C. et al. VeriHarness: Scaling Agentic Verification for Long-Horizon Tasks. Preprint at https://arxiv.org/abs/2610.00972 (2026).
43. Bobadilla-Suarez, S., Suh, B. & Fortin, R. Audit the Scaffold, Not the Checkpoint: A Stationarity Dichotomy for Recursive Self-Improvement in Agentic Coding. Preprint at https://arxiv.org/abs/2609.34924 (2026).
44. Snell, C., Lee, J., Xu, K. & Kumar, A. Scaling LLM Test-Time Compute Optimally can be More Effective than Scaling Model Parameters. In *Proc. International Conference on Learning Representations* (2025); https://arxiv.org/abs/2408.03314.
45. Shinn, N. et al. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems* **36** (2023); https://arxiv.org/abs/2303.11366.
46. Srinivasan, V. A Methodology for Selecting and Composing Runtime Architecture Patterns for Production LLM Agents. Preprint at https://arxiv.org/abs/2605.20173 (2026).
47. DeepSeek. Models & Pricing. DeepSeek API Docs, https://api-docs.deepseek.com/quick_start/pricing (2026).
48. Liu, T. et al. Budget-Aware Tool Use Enables Effective Agent Scaling. Preprint at https://arxiv.org/abs/2511.17006 (2025).
49. Zhou, K. et al. Beyond Max Tokens: Stealthy Resource Amplification via Tool Calling Chains in LLM Agents. Preprint at https://arxiv.org/abs/2601.10955 (2026).
50. Duan, D. et al. How Much Can Tool Calls Reveal About You? Privacy Leakage from Tool-Call Traces. Preprint at https://arxiv.org/abs/2610.41885 (2026).
51. Greshake, K. et al. Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. Preprint at https://arxiv.org/abs/2302.12173 (2023).
52. Debenedetti, E. et al. AgentDojo: A Dynamic Environment to Evaluate Prompt Injection Attacks and Defenses for LLM Agents. In *Advances in Neural Information Processing Systems* **37** (Datasets and Benchmarks Track) (2024); https://arxiv.org/abs/2406.13352.
53. Longpre, S. et al. Consent in Crisis: The Rapid Decline of the AI Data Commons. Preprint at https://arxiv.org/abs/2407.14933 (2024).
54. Roth, E. The biggest names in AI are backing a new way to opt out of scraping. The Verge, https://www.theverge.com/tech/1023763/openai-google-microsoft-ai-scraping-standard-opt-out (2026).
55. Mueller, J. Supporting a shared AI training control for websites. Google Search Central Blog, https://developers.google.com/search/blog/2026/10/ai-training-control (2026).
56. Chowdhury, R. terms.txt: A Consent and Compensation Protocol for Agentic Web Access. Preprint at https://arxiv.org/abs/2609.11152 (2026).
57. Song, Y. et al. Training Foundation Models with a Single Algorithm. Preprint at https://arxiv.org/abs/2610.40564 (2026).
58. Yu, Q., Ma, X., Bin, X. & He, D. Your LR Schedule May Not Be Principled: Effective Learning Rate Scheduling in Weight-Normalized Training. Preprint at https://arxiv.org/abs/2610.40362 (2026).
59. Liu, Z. et al. Effective Learning Rate Governs Loss Dynamics in Language Model Pretraining. Preprint at https://arxiv.org/abs/2608.24814 (2026).
60. Liu, F., Liu, Y., Sun, R., Luo, L. & Guo, S. Coding Agents Have Converged: Why the SWE-bench Leaderboard Can No Longer Order Its Top Entries, and What to Measure Instead. Preprint at https://arxiv.org/abs/2609.17394 (2026).
61. McNemar, Q. Note on the Sampling Error of the Difference Between Correlated Proportions or Percentages. *Psychometrika* **12**, 153–157 (1947); https://doi.org/10.1007/BF02295996.
62. Dietterich, T. G. Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Comput.* **10**, 1895–1923 (1998); https://doi.org/10.1162/089976698300017197.
63. 弘犀智能（Great Unicorn Intelligence）. 企业智能体平台 · Harness（Enterprise Agent Platform · Harness）. https://www.hxallai.com/en/platform/agent/ (2026).

Data availability

No new experimental data were generated. Numbers quoted in the text come from the listed references and official documentation; the data used in the illustrative calculation are generated entirely by script under the assumptions stated in the text.

Code availability

The script for the illustrative calculation and Figs. 1–3 is src/figs.py (Python, fixed random seed), provided with the article package; it reproduces every number in Table 4 and Figs. 1–3.

Competing interests

The authors are employees of Great Unicorn Intelligence Technology (Shenzhen) Co., Ltd. The Enterprise Agent Platform Harness mentioned in the text is a product of the company.

Copyright Notice: © 2026 Great Unicorn Intelligence Technology (Shenzhen) Co., Ltd. All rights reserved. Any reposting, quotation in commentary, or adaptation must credit the author (Great Unicorn Intelligence Research Team) and cite the source (Great Unicorn Intelligence, https://www.hxallai.com).