TITRATE: EVALUATING LLM AGENTS WITH FEWER EXECUTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Agentic benchmarks run an agent through multi-step tasks in real environments. Each run costs money and compute, and its outcome is random. Existing efficient-evaluation methods fix a budget, typically 10% of the tasks, and report whatever accuracy results. An evaluator usually needs the opposite: given a target accuracy, how many runs are required? With 10% of the tasks, the share of agents scored within four points of their full-benchmark score ranges from 39% to 83% across five benchmarks, so the same budget buys very different reliability. We propose TITRATE, which derives the number of runs from the tolerance and adapts item response theory to agentic benchmarks in three ways. Task parameters are fitted on the few dozen agents a benchmark has already evaluated, so TITRATE shrinks the estimated discriminations to keep tasks that look informative by chance from being selected first. An agent–task random effect separates an agent's lasting advantage on a task from run-to-run randomness, which tells TITRATE which tasks are worth rerunning. Because the model's own error estimate is optimistic, TITRATE scales it by a factor learned by replaying the procedure on those agents, which turns a tolerance into a task count. Asked to place 95% of agents within four points of their full-benchmark score, TITRATE places 91% to 97% on all five benchmarks under random splits. With 10% of the tasks, it estimates the full-benchmark score at least as accurately as six existing methods on every benchmark, and choosing which tasks to rerun matches uniform rerunning with about 20% fewer runs.

1 INTRODUCTION

LLM-based agents have become increasingly capable of solving long-horizon tasks through iterative interactions with tools and environments (Yang et al., 2024; Wang et al., 2025b). This progress has driven a shift from conventional benchmarks to agentic benchmarks that assess the completion of an entire complex task (Yao et al., 2024; Siegel et al., 2024; The Terminal-Bench Team, 2025). For example, in SWE-bench (Jimenez et al., 2023), an agent must navigate a repository, execute commands, modify code, and respond to test feedback to resolve a real issue.

Evaluations on agentic benchmarks are costly and time-consuming. A single pass over SWE-bench Verified costs $760 to $935 and requires more than a day of sequential machine time (The OpenHands Team, 2026). This cost is amplified as evaluation moves from one-time leaderboard reporting into the optimization loop of automated harness tuning and self-evolving agents. Such systems repeatedly propose changes and use benchmark performance to decide which should survive. Running the full benchmark for every candidate is prohibitively expensive, yet an inaccurate estimate can promote a harmful change or discard a genuine improvement. The resulting challenge is therefore not merely to approximate the full-benchmark score cheaply, but to collect just enough evidence to make each optimization decision reliably.

Existing methods address this challenge by predicting an agent's full-benchmark score from a subset whose size is fixed in advance (Polo et al., 2024; Kipnis et al., 2024; Duan et al., 2026). However, a fixed budget may waste executions after sufficient evidence has been collected or stop before the estimate becomes reliable. Although adaptive methods such as ATLAS (Li et al., 2025) have begun to address this limitation, reliably determining when to stop for different agents and benchmarks remains an open challenge.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

These methods were built for language-model benchmarks: thousands of cheap single-response items, and hundreds to thousands of models to fit them on. Agentic evaluation differs in three ways. Each shapes one part of our design. First, the stopping rule should be stated in score points. FLUID (Hofmann et al., 2025) and ATLAS (Li et al., 2025) stop when the standard error of the estimated ability falls below a threshold. A leaderboard reports a pass rate, and a precise ability is not a precise pass rate. At the tightest threshold in its paper, ATLAS puts 73% of agents within four points of their full-benchmark score on SWE-bench Verified and 96% on CORE-bench Hard. An evaluator cannot choose a threshold for the accuracy they want. Second, the same task can give a different result on the next run (Mustahsan et al., 2025). For a language model, one observation is a single response, and repeating it is cheap (Zhou et al., 2026; Wu et al., 2026). For an agent, it is a container run of minutes to an hour. Part of the variation is random. Part is a persistent match between one agent, with its tools, prompts and workflow, and one task. On Terminal-Bench 2.0, an agent that passed a task passed the next attempt 82% of the time; one that failed passed 15% of the time. A model of independent attempts predicts 75% and 21%. An evaluator has to separate the two sources and decide which tasks to run again. No subset-based method does either. Third, few agents are available to fit the task parameters. Only agents a leaderboard has already run can be used: 10 to 130 here, against 102 models for FLUID and 3,467 to 4,680 for ATLAS. With so few, some tasks separate them by chance and receive a high discrimination. Maximum-information selection then picks these tasks first, the ones whose estimated informativeness is least trustworthy (van der Linden & Glas, 2000).

To fill these gaps, we propose TITRATE, a sequential evaluator that converts a requested score tolerance into a number of executions. At each step, it selects a new task, updates the estimated full-benchmark score, and stops when a calibrated error criterion meets the tolerance. TITRATE regularizes task parameters through validation-selected shrinkage, calibrates its stopping rule by replaying the procedure on historical agents, and, where attempts repeat, runs the bank once and then reruns the tasks where a further attempt most reduces score uncertainty. To our knowledge, it is the first sequential evaluator for agentic benchmarks that adaptively allocates repeated attempts.

Across five benchmarks and six baselines, TITRATE has the lowest score error at one tenth of the tasks, or is statistically tied with the best baseline. For the stopping rule, the evaluator sets a tolerance and the share of agents that should fall within it. Asked for 95% of agents to be within four points of their full-benchmark score, the rule places 91% to 97% of held-out agents within four points, running 30 to 112 tasks per agent on average depending on the benchmark. These shares are measured after the fact rather than promised in advance, so we also give a variant that comes with a guarantee: if a new agent comes from the same population as the agents used to calibrate the rule, and none of them was used to fit the task parameters, the probability that its error exceeds the tolerance is at most the level the evaluator states. This variant is built with conformal risk control (Angelopoulos et al., 2024) and runs more tasks than the default rule. When every baseline is calibrated in the same way to give the same guarantee, TITRATE needs the fewest tasks on all four benchmarks with one run per task. On Terminal-Bench 2.0, choosing which tasks to rerun reaches the same score error as rerunning all tasks equally with 20% fewer runs.

In summary, our contributions are as follows:

- We propose TITRATE, a sequential evaluator that turns a requested score tolerance into a number of executions. It shrinks task parameters with a validation-selected strength, calibrates its stopping rule by replaying the procedure on historical agents, and chooses which tasks to rerun when attempts repeat. To our knowledge, it is the first sequential evaluator for agentic benchmarks that allocates repeated attempts adaptively.

- We give a variant of the stopping rule built with conformal risk control. For a new agent from the same population as the calibration agents, it bounds the probability that the score error exceeds the tolerance by a level the evaluator sets.

- We evaluate TITRATE on five agentic benchmarks against six baselines. It has the lowest score error at one tenth of the tasks or is statistically tied with the best baseline, needs the fewest tasks under a shared conformal guarantee on all four single-run benchmarks, and on Terminal-Bench 2.0 matches the error of uniform reruns with 20% fewer runs.

2 Background and Related Work

Item response theory and adaptive testing. The two-parameter logistic (2PL) model of item response theory (Lord, 1980; Embretson & Reise, 2000) gives agent i an ability θ_i and task j a difficulty b_j and a discrimination $a_j > 0$. The probability of a success and the Fisher information of one response about θ are

$$p_j(\theta) = \text{sigmoid}(a_j(\theta - b_j)), \qquad I_j(\theta) = a_j^2\, p_j(\theta)\big(1 - p_j(\theta)\big). \tag{1}$$

Task parameters are fitted to agents that have already completed the benchmark and are then held fixed. Computerized adaptive testing runs the unused task with the largest $I_j(\hat{\theta}_i)$, updates $\hat{\theta}_i$, and conventionally stops when the standard error of the ability, $\big(\sum_{j \in Q_i} I_j(\hat{\theta}_i)\big)^{-1/2}$ over the tasks Q_i run so far, falls below a threshold (Wainer et al., 2000). That quantity is computed from the fitted parameters as if they were known, and it is expressed on the ability scale, not in score points (Appendix A).

Efficient evaluation, and what it assumes. tinyBenchmarks and metabench learn compact shared evaluations (Polo et al., 2024; Kipnis et al., 2024), and Growing Pains calibrates new datasets with earlier item parameters held fixed, on several hundred language models (Habba et al., 2026); its anchors are the same for every model, and how many runs one agent needs is not asked. A mid-range difficulty filter keeps tasks of intermediate historical pass rate (Ndzomga, 2026); we include it as a panel. FLUID and ATLAS use IRT to select tasks adaptively (Hofmann et al., 2025; Li et al., 2025). Their evaluations still fix the size: FLUID sweeps subsets of 10 to 500 items and demonstrates a standard-error stopping rule on one model and one benchmark, and ATLAS stops on the standard error of the ability and reports its deviation from the whole-bank ability; PTA-IRT reports its main comparison at a tenth of the bank (Duan et al., 2026); Truong et al. (2025) predict item difficulty from content. A separate line stops once an interval around the score is narrow enough, without a model of the tasks (Wang et al., 2025a; Zhou et al., 2026; Hsu & Shekhar, 2026; Arviv et al., 2026).

Agent-specific information and repeated attempts. Agent Psychometrics predicts task success from task features and model–scaffold ability components (Ge et al., 2026), and PTA-IRT uses historical trajectories for training subset selection and ability estimation (Duan et al., 2026). The random effect we use for repeated attempts has psychometric precedents in generalizability theory and in testlet and random-item IRT models (Cronbach et al., 1972; Brennan, 2001; Bradlow et al., 1999; De Boeck, 2008); our contribution is their use for allocating agent reruns (Appendix E).

3 Method

Problem. A benchmark is a bank of N tasks. A *run* of agent i on task j returns $Y_{ijr} \in \{0, 1\}$, where r indexes the T_{ij} attempts the benchmark records, and the benchmark ranks agents by the pass rate

$$s_i = \frac{1}{N} \sum_{j=1}^{N} \frac{1}{T_{ij}} \sum_{r=1}^{T_{ij}} Y_{ijr}. \tag{2}$$

The task is to estimate s_i from a subset of the runs. The procedure deals with two kinds of agents. *Training agents* have already run the whole benchmark, so their full records are available. In practice they are earlier submissions to a leaderboard. We use them to fit the task parameters and to calibrate the stopping rule. A *test agent* is a new agent to be evaluated. The cost of an evaluation is measured by the number of runs made on the test agent (Appendix D.8) On our benchmarks $T_{ij} = 1$, except on Terminal-Bench 2.0, which records several attempts per task. The evaluator states a *tolerance* ε in percentage points and a *reliability* $1 - \alpha$: the error $|\hat{s}_i - s_i|$ should be at most ε for a share $1 - \alpha$ of test agents. We report the share actually reached on held-out test agents. To rank a cohort, a *certificate* fixes one task count from the training agents before any test agent runs; it carries no finite-sample guarantee.

3.1 Overview

TITRATE has one training stage and two protocols that use it (Figure 1). Training fits an item response model to the training agents (Section 3.2) and calibrates the stopping rule by replaying the procedure

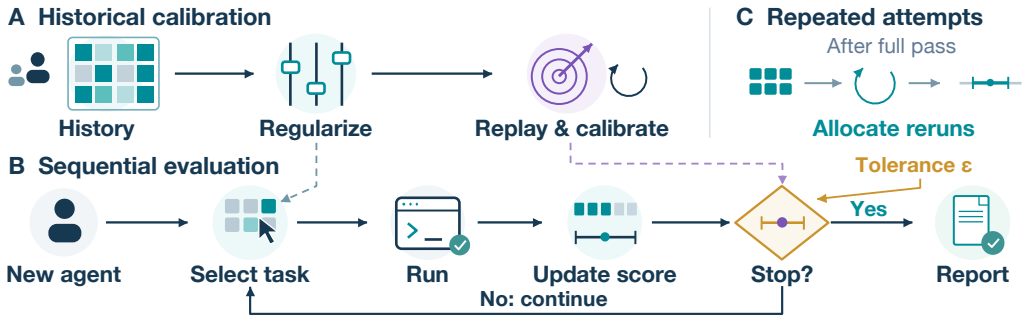

Figure 1: **TITRATE overview. (A)** Historical evaluations fit a regularized task model and calibrate the stopping rule through held-out replay. **(B)** Each new agent is evaluated one execution at a time, selecting tasks and updating its estimated full-benchmark score until the calibrated stopping criterion meets tolerance ε. **(C)** When repeated attempts are allowed, after a complete first pass, reruns target the task with the greatest expected reduction in score uncertainty.

on them (Section 3.4). The *per-agent protocol* evaluates one test agent step by step: select a task, run it, update the posterior over the agent's ability, re-estimate the score and its error, and stop once the calibrated error is at most ε (Sections 3.3 and 3.4). The number of tasks is not fixed in advance. It depends on the outcomes and differs across agents. On benchmarks that allow repeated attempts, the agent first runs every task once and then reruns chosen tasks (Section 3.5). The *cohort protocol* ranks a group of new agents rather than scoring each one. Before any of them runs, it uses the training agents to choose one task count K^* at which the ranking is expected to reach a target Kendall τ^*, and every agent then receives K^* adaptively chosen tasks (Section 3.6).

The *uncertainty* is the posterior standard deviation of the quantity it estimates, i.e., how far the model believes the true value may lie from its estimate. We keep three kinds of uncertainty apart: (i) the posterior uncertainty of the ability θ_i, (ii) the posterior uncertainty of the predicted score $\hat{s}_i$, and (iii) the actual error $|\hat{s}_i - s_i|$ on agents the model has not seen. Prior adaptive-testing rules stop on (i), which is not even in score points. The per-agent protocol stops on (iii). Since (iii) cannot be observed, it is estimated from (ii), scaled by a factor learned on the training agents.

3.2 FITTING TASK PARAMETERS WITH FEW TRAINING AGENTS

We use the 2PL model of Eq. 1. With only a few dozen training agents, a task can separate them well by chance and receive a large discrimination a_j. Since information grows with a_j^2, maximum-information selection then picks exactly these tasks first, although their parameters are the least reliable (van der Linden & Glas, 2000). We therefore fit the model with a penalty $\lambda \sum_j (a_j - 1)^2$ that pulls every discrimination toward 1, and cap a_j to $[0.4, 2.5]$. The weight λ is chosen by validation on internal splits of the training agents. Notably, no test agent's responses are used.

The fitted parameters serve two uses, so we fit the model twice, with a different λ for each. λ_{select} gives the parameters used to choose the next task; we pick it by how reliably the certificate built from it meets its target on the internal splits. λ_{score} gives the parameters used to predict the unrun tasks in Eq. 3, and hence the error the stopping rule reads; we pick it by held-out log likelihood on the same splits. One choice does not serve both uses. Picking λ_{select} by log likelihood lowers the certificate's worst realized reliability from 0.77 to 0.43, while using the certificate's choice for both raises the score error on SWE-bench Verified by 0.33 points (Appendix D.13). Grids, priors and other details are in Appendix A.

3.3 SEQUENTIAL EVALUATION OF A TEST AGENT

Algorithm 1 in the appendix states the loop; this section describes it. The test agent's ability is a posterior over a grid, started from a normal prior fitted to the training agents. Each step runs the unrun task with the largest $I_j(\hat{\theta}_i)$ at the posterior mean and multiplies the posterior by the likelihood of the outcome, so a strong agent is steered toward harder tasks and a weak one toward easier tasks,

and two agents rarely see the same tasks. With observed tasks Q_i the score estimate combines what was seen with what the model predicts for the rest,

$$\hat{s}_i = \frac{1}{N}\Big(\sum_{j\in Q_i} Y_{ij1} + \sum_{j\notin Q_i} p_j(\hat{\theta}_i)\Big). \tag{3}$$

3.4 Calibrated stopping for a score target

After K tasks the posterior over θ_i induces a posterior over $\hat{s}_i$, whose variance sums, over the unrun tasks, the Bernoulli variance $p_j(1-p_j)$ and the variance of their predicted mean under the posterior of θ_i; the predicted absolute error is $u_i(K) = \sqrt{2/\pi}\ \mathrm{SD}_{\mathrm{post}}(\hat{s}_i)$. This prediction is optimistic: it treats task parameters estimated from a few dozen agents as known.

We correct it by replay. Inside the training pool, one part plays history and the other plays test agents; Algorithm 1 runs on the latter without its stopping test, recording at every budget the predicted error u and the realized error $e = |\hat{s}_i(K) - s_i|$ against that agent's known score. How far the model understates its error depends on how confident it currently is, so the pairs are cut into ten bins at the deciles of u, and the $(1-\alpha)$-quantile of e/u within a bin gives a factor $c_{1-\alpha}(u)$: when the model claims u, the realized error stayed below $c_{1-\alpha}(u)\,u$ in a share $1-\alpha$ of the replays. A bin with fewer than 20 pairs takes the pooled quantile. We do not force $c_{1-\alpha}$ to be monotone in u (Appendix A.2). The rule stops at the first K with

$$c_{1-\alpha}\big(u_i(K)\big)\,u_i(K) \le \varepsilon. \tag{4}$$

Stopping is optional, and the first moment a model looks confident is selected for optimism, so the replay also records the signed error at each internal agent's own stopping count and subtracts its mean from the reported score. We use $\alpha = 0.05$. This rule is empirical, and we measure its realized reliability on held-out agents.

A guaranteed variant replaces the binned factor by one multiplier c, stopping when $c\,u_i(K) \le \varepsilon$, and sets c by conformal risk control (Angelopoulos et al., 2024). Each calibration agent is replayed under c; its loss is the indicator that its error at its own stopping time exceeds ε; and c is the smallest value whose corrected empirical risk is at most α. The stopping time sits inside the loss, as in risk-controlled early exiting (Ringel et al., 2024; Jazbec et al., 2024), so optional stopping does not void the statement: for exchangeable agents that the item fit has not seen, the error exceeds ε with probability at most α (Appendix C.1.3).

3.5 Extension to repeated attempts

Where the protocol repeats attempts, a step may rerun a task, and the model must say what a rerun can reveal. Attempts of one agent on one task are not independent: beyond run-to-run noise, an agent and a task have a persistent match, a tool the agent lacks or a strategy it happens to use. A random effect shared by all attempts in a cell represents it,

$$\operatorname{logit} p_{ij} = a_j(\theta_i - b_j) + \varepsilon_{ij}, \quad \varepsilon_{ij} \sim \mathcal{N}(0, \sigma_j^2), \quad Y_{ijr} \mid p_{ij} \sim \mathrm{Bernoulli}(p_{ij}). \tag{5}$$

Conditional on p_{ij} the attempts are independent; once ε_{ij} is integrated out they are dependent, so a first failure lowers the posterior of the cell and a first success raises it. The scales σ_j come from the training agents' repeated attempts; with one attempt per cell we set $\sigma_j = 0$, recovering Equation 1.

The random effect bounds what a rerun can buy. With $\bar{p} = \mathbb{E}[p_{ij}]$ and $\rho = \mathrm{Var}(p_{ij})/[\bar{p}(1-\bar{p})]$ over the random effect, the mean of R attempts has variance

$$\mathrm{Var}(\bar{Y}_{ij}) = \frac{\bar{p}(1-\bar{p})}{R}\big\{1 + (R-1)\rho\big\}, \tag{6}$$

which tends to $\bar{p}(1-\bar{p})\rho$ rather than to zero. A model with $\rho = 0$ overstates the value of every additional attempt.

Allocating reruns. The score estimate becomes the observed successes plus the posterior expected successes of the attempts not yet bought, and its variance is approximated cell by cell. One more attempt in cell (i, j) is worth the expected reduction of that variance,

$$g_{ij} = V_{ij} - \big[\,\hat{p}_{ij}\,V_{ij}^{(1)} + (1-\hat{p}_{ij})\,V_{ij}^{(0)}\big], \tag{7}$$

Benchmark	Domain	Agents	Tasks
SWE-bench Verified	code repair	174	500
SWE-bench Lite	code repair	84	300
CORE-bench Hard	scientific reproduction	66	45
τ-bench airline	tool use	47	50
Terminal-Bench 2.0	terminal tasks	127	89

Table 1: **The five benchmarks.**

with $\hat{p}_{ij}$ the predictive probability of a success and $V_{ij}^{(y)}$ the variance after outcome y. Reruns go to the cells with the largest g_{ij}, in batches, after one pass over the bank. A misfit check guards this allocation. After the first pass it compares the test agent's mean log marginal likelihood with the fifth percentile of held-out training agents; below it, half of every later batch follows a uniform order (Appendices A.1 and C.2.3).

Stopping on runs is a weaker rule than Equation 4 and we keep the two apart: it carries no replay margin, stopping the first time $1.645 \cdot \mathrm{SD}_{\mathrm{post}}(\hat{s}_i) \le \varepsilon$ under the repeat-aware posterior, so it states no reliability and we report only the share of agents it leaves within ε.

3.6 Prospective budgeting for ranking a cohort

The cohort protocol fixes one task count K^* for all test agents in advance, so that every agent in the cohort is measured on the same number of tasks. Replay on the training agents gives $v(K)$, the variance of the ability error after K tasks, and V is the variance of the training agents' abilities; under a Gaussian approximation the expected Kendall correlation between estimated and true ability is $\frac{2}{\pi}\arcsin$ of their correlation, so

$$\widehat{\tau}(K) = \frac{2}{\pi}\arcsin\sqrt{\frac{V}{V+v(K)}} + d, \qquad K^* = \min\bigl\{K \ge 4 : \widehat{\tau}(K) - m_q(K) \ge \tau^*\bigr\}, \quad (8)$$

where the offset d and the safety margin $m_q(K)$ are also estimated by replay (Appendix A). If no K qualifies, the rule returns the full bank and marks the target uncertified.

4 Experimental Setup

We evaluate TITRATE across five benchmarks (Table 1): SWE-bench Verified and Lite for Github issue resolving, CORE-bench Hard for scientific reproduction, τ-bench airline for tool use, and Terminal-Bench 2.0 for terminal tasks (Jimenez et al., 2023; Siegel et al., 2024; Yao et al., 2024; The Terminal-Bench Team, 2025). Their records come from the public leaderboards and from two released collections of agent responses (Kapoor et al., 2025; Ge et al., 2026); only Terminal-Bench 2.0 records repeated attempts.

We use 30 random splits, assigning 75% of the agents to training and the remaining 25% to testing. Every method sees the same splits, so all comparisons are paired.

The tolerance is the evaluator's input and nothing in the method depends on its value; we report $\varepsilon = 4$, 3 and 2 points, which span what these banks can deliver.

Two settings are evaluated. In the single-agent setting the question is one test agent's own full-bank score, the metric is the error of the prediction, and the stopping rule decides the budget. In the cohort setting the held-out agents of a split are ranked together, the metrics are Kendall τ and Spearman ρ against the official order, and the certificate fixes one task count in advance.

We compare against the released implementations of tinyBenchmarks, FLUID, ATLAS and PTA-IRT (Polo et al., 2024; Hofmann et al., 2025; Li et al., 2025; Duan et al., 2026), and against two static panels: a classical panel of the most discriminating tasks within five difficulty strata, and the mid-range panel of Ndzomga (2026). For repeated attempts we compare uniform reruns and allocation under a binomial 2PL. Following Duan et al. (2026), every method predicts each test agent's full-bank pass rate from the tasks it ran; in Tables C.25 and C.26, MAE, τ and ρ all come from this one

predicted score per method, while Table 4 ranks each baseline by the ranking output of its released adaptive script (Table B.2).

Because every method sees the same splits, uncertainty comes from a paired bootstrap over them, with 2000 resamples. One family of comparisons is corrected for multiplicity by the Holm procedure: the 36 score-error comparisons of TITRATE against ATLAS and against random subsets, on the four single-attempt banks. Intervals elsewhere are unadjusted, and a difference that is not significant is not evidence of equivalence. Appendix B records the choices we made after seeing results, and Appendix B.4 the compute: fitting and calibrating a benchmark takes 1.5 to 18 minutes per split on one CPU core, against one to five hours for ATLAS's released pipeline.

5 RESULTS

5.1 EVALUATING ONE AGENT TO A SCORE TOLERANCE

Given the tolerance instead, the rule spends what that tolerance needs. At four points it stops after 112 tasks on Verified, 63 on Lite, 30 on CORE-bench Hard, 41 on τ-bench airline and 64 on Terminal-Bench 2.0, and 0.91 to 0.97 of test agents finish within the tolerance, against the 0.95 the rule states (Table 2).

Table 3 sets three rules an evaluator can actually follow against one another. A tenth of the bank is the convention: the count is known in advance and the reliability is whatever follows, 0.39 to 0.83. ATLAS is the only baseline whose released code decides its own budget, and we give it the tightest threshold in its paper, the one that serves it best. On four of the five banks TITRATE is the cheaper rule, at least as reliable on three of them and 0.01 below on CORE-bench Hard: on SWE-bench Lite it spends 63 tasks for 0.94 where ATLAS spends 104 for 0.82, and on τ-bench airline 41 for 0.97 against 46 for 0.93. SWE-bench Verified is the exception: ATLAS stops after 48 tasks and leaves 0.73 of agents within four points, while TITRATE spends 112 and reaches 0.94; ATLAS is at the tightest threshold its paper offers, and because that threshold is written in ability no setting of it states a score reliability; Table C.3 gives it that calibration directly. A count fixed in advance at the rule's own mean meets the target about as often (the fixed column of Table 2), so what the rule adds is not a smaller bill but a budget that follows from a stated tolerance. Granting every baseline the budget it would have chosen in hindsight, our prospective count is at or below it in 33 of the 39 cells where both budgets are determined; in the rest the baseline had not met the criterion at the last budget its code was run at (Table D.8).

Asking for a guarantee instead of a measured share costs tasks but does not cost the comparison. Calibrating the margin by conformal risk control on held-out agents gives the rule a finite-sample statement, and applying the same calibration to each baseline's task count gives every method that statement; among the baselines that reach it, TITRATE needs the fewest tasks in all twelve bank–tolerance cells, and a count fixed in advance under the same calibration is within a tenth of it either way (Table C.3).

Where the protocol allows repeated attempts, a weaker rule counts runs and still saves them. A two-point target is out of reach on Terminal-Bench 2.0 if each task is run once. Allowing reruns and stopping on the posterior standard deviation of the official score, without the replay margin and so without a stated reliability, the procedure reaches 3, 2 and 1 points with 131, 183 and 266 runs, leaves 0.90, 0.89 and 0.84 of agents on target, and uses 23 to 26% fewer runs than uniform rerunning needs for the same share (Table C.21 in the appendix).

Score accuracy holds when later agents are predicted from earlier ones, the order in which a benchmark actually receives them. With training agents taken from the earliest submissions and test agents from the next quarter, over six rolling cutoffs, our score error at a tenth of the bank is 2.31 points on Verified and 3.67 on Lite against 3.90 and 6.42 for a random subset, lower in all 24 benchmark–budget–cutoff cells, and the tasks needed for a three-point error are 20, 42 and 15 on Verified, Lite and Terminal-Bench 2.0 against 47, 72 and 22 for ATLAS (Appendix C.1.7).

	$\varepsilon = 4$ points			$\varepsilon = 3$ points			$\varepsilon = 2$ points		
Bank (tasks)	tasks	met	fixed	tasks	met	fixed	tasks	met	fixed
Verified (500)	112	0.94	0.92	170	0.93	0.90	191	0.81	0.79
Lite (300)	63	0.94	0.95	105	0.94	0.93	148	0.94	0.93
CORE (45)	30	0.95	0.93	33	0.95	0.93	38	0.97	0.94
τ-bench (50)	41	0.97	0.96	44	0.97	0.96	45	0.98	0.96
TB 2.0 (89)	64	0.91	0.91	83	0.83	0.83	*all*	0.62	0.61

Table 2: **What the stopping rule spends and delivers, counted in tasks.** *tasks*: mean stopping count; *met*: share of agents within ε, against a stated 0.95; *fixed*: the same share at a count fixed in advance at that mean; *all*: the bank is exhausted first; on Verified and Lite the rule runs to at most 200 tasks. On Terminal-Bench 2.0 a task consumes its five recorded attempts. Mean over 30 splits.

	A tenth of the bank		ATLAS, $\mathrm{SE}(\hat{\theta}) \leq 0.1$		TITRATE, $\varepsilon = 4$ points	
Bank (tasks)	tasks	met	tasks	met	tasks	met
Verified (500)	50	0.83	48	0.73	112	**0.94**
Lite (300)	30	0.80	104	0.82	**63**	**0.94**
CORE (45)	4	0.39	40	**0.96**	**30**	0.95
τ-bench (50)	5	0.40	46	0.93	**41**	**0.97**
TB 2.0 (89)	9	0.53	74	0.90	**64**	**0.91**

Table 3: **Three rules an evaluator can actually follow, at a four-point tolerance.** *met*: share of test agents within four percentage points of their full-bank score. The tenth is fixed in advance; both rules are calibrated before the test agent runs and stop on its outcomes. ATLAS is given the tightest threshold in its paper, the one that serves it best. TITRATE states 0.95. Bold: the highest reliability in the row, and TITRATE's count where it is below ATLAS's.

5.2 DECIDING WHICH TASKS TO RUN AGAIN

Choosing which cells to rerun saves runs, and the saving does not come from the change of estimator. Holding the first pass over the 89 tasks fixed and varying only the later allocation, uniform reruns reach 0.87 points of error at 267 runs, while our allocation reaches 0.81 at 214: a paired difference of -0.06 points with a 95% interval of $[-0.11, -0.01]$, or a fifth fewer runs for the same accuracy (Table C.16 in the appendix). Charging each run its token cost, which differs 34-fold across tasks, improves the allocation further at every budget and costs no ranking quality; the misfit check that guards it was fixed on Terminal-Bench 2.0 and confirmed on release 4.0 (Appendix D.12).

The saving comes from the interaction model, which predicts a repeated attempt where an independence model cannot. After an observed pass the next attempt succeeds 0.823 of the time and after a failure 0.151; the binomial 2PL predicts 0.752 and 0.212, and TITRATE 0.818 and 0.156, with a higher held-out log likelihood on all 30 splits. After one pass the same posterior predicts pass@5 and pass^5 with a mean signed error of $+0.9$ and -0.4 points, where the binomial model is off by $+6.6$ and -6.6; the mean absolute errors are 2.9 and 3.2 points against 6.7 and 6.9 (Appendix C.2.4).

Scored against three held-out attempts per cell that no method may acquire, TITRATE improves on a complete single-attempt pass by 0.48 points at N runs on Terminal-Bench 2.0, significantly; on two smaller releases the difference is not, and thinning the pool places the boundary between 13 and 17 training agents (Table C.17 in the appendix, Appendix D.7).

5.3 RANKING A COHORT OF AGENTS

A target ranking costs the fewest tasks under TITRATE, in every benchmark–target cell of Table 4, significantly fewer than a baseline in 45 of 60 comparisons over four targets and never significantly more (Table C.4 adds the two lower targets and a random subset). The gap widens with the target: $\tau = 0.90$ costs 45 tasks on SWE-bench Verified against 60 for FLUID and 84 for ATLAS, and 48 on Lite against 84 and 107; at $\tau = 0.95$ FLUID needs 300 tasks on Verified and ATLAS does not reach it within the bank, and on τ-bench airline neither reaches even 0.80.

Benchmark	Target	FLUID	ATLAS	TITRATE
SWE-bench Verified	$\tau \geq 0.9$	60 ± 17	$84 \pm 16^{\dagger}$	$\mathbf{45 \pm 13}$
	$\tau \geq 0.95$	$300 \pm 44^{\dagger}$	n.r.†	$\mathbf{188 \pm 22}$
SWE-bench Lite	$\tau \geq 0.9$	$84 \pm 24^{\dagger}$	$107 \pm 47^{\dagger}$	$\mathbf{48 \pm 6}$
	$\tau \geq 0.95$	250 [188, n.r.]†	n.r.†	$\mathbf{107 \pm 26}$
CORE-bench Hard	$\tau \geq 0.9$	$28 \pm 2^{\dagger}$	n.r.†	$\mathbf{23 \pm 1}$
	$\tau \geq 0.95$	n.r.†	n.r.†	$\mathbf{32 \pm 1}$
τ-bench airline	$\tau \geq 0.9$	n.r.†	n.r.†	$\mathbf{34 \pm 3}$
	$\tau \geq 0.95$	n.r.†	n.r.†	$\mathbf{43 \pm 1}$
Terminal-Bench 2.0	$\tau \geq 0.9$	$20 \pm 3^{\dagger}$	18 ± 4	$\mathbf{16 \pm 2}$
	$\tau \geq 0.95$	n.r.	n.r.	**67 [48, n.r.]**

Table 4: **Tasks per test agent needed to reach a target ranking quality.** First budget at which each method's mean Kendall τ over the 30 splits reaches the target, interpolated between budgets. $\pm$: half the width of the 95% interval from resampling splits; bold: fewest; n.r.: the mean curve never reaches the target on the whole bank; †: TITRATE needs significantly fewer tasks. The target is the official full-bank pass-rate order.

Stating 0.98 for the cohort certificate, the realized reliability averages 0.929 over 20 benchmark–target settings and clears 0.90 in 18 of them, for 11% more tasks than stating 0.95 (Appendix C.1.8).

6 WHY THE METHOD HELPS, AND WHAT CARRIES THE CLAIMS

Within the IRT family, the gain comes from stable item calibration: swapping our item parameters for FLUID's costs up to 0.101 in Kendall τ while swapping the ability estimator costs at most 0.006 (Table D.1). The fitted discriminations show why: on τ-bench airline their 95th percentile is 1.23 under our penalized fit, 2.72 under FLUID's and 55.3 under ATLAS's. This is capitalization on calibration error (van der Linden & Glas, 2000): a method built for a hundred or more models loses ground on a leaderboard of dozens.

7 LIMITATIONS

The default stopping rule is empirical: its margin is a quantile of realized over predicted error on replayed training agents, and its reliability is a share measured over held-out agents. The guaranteed variant restores a finite-sample statement, but it sets agents aside from the item fit, it spends more tasks, and its statement is marginal over exchangeable agents rather than conditional on one. Both rest on training agents that represent the agent being evaluated: calibrated on earlier agents, the per-agent rule realizes 0.85–0.88 of a stated 0.95 on later ones and the cohort certificate meets its target in one to four of six cutoffs, and recalibrating on recent agents alone does not restore the level (Appendix C.1.7). Reruns are tested on three releases of one benchmark family, and we did not simulate them on the others. The 2PL does not model guessing; a 3PL fitted the same way agrees with it to within 0.011 (Appendix D.1). The savings are measured in runs. Dollars are measured only for 35 single-scaffold SWE-bench Verified submissions that report per-task cost: for them the four-point rule pays 22% of the full-bank bill, $43 against $195 per agent (Appendices F and D.9).

8 CONCLUSION

Treating agentic evaluation as target-driven sequential measurement changes what an evaluator has to decide: not which fraction of the bank to buy, but how accurate the answer must be. In TITRATE, three components built for agentic data hold a stated reliability of 0.95 to a realized 0.91–0.97 at four points and match uniform reruns with a fifth fewer executions.

Reproducibility statement

The experiments use public benchmark records. Appendix B.3 describes the response matrices; Appendix B.2 records baseline implementations and modifications. All comparisons use shared agent splits. Appendix B.6 reports the development log and adoption criteria. The supplementary material contains the code, the experiment scripts and the result files from which every table is generated.

Ethics statement

The study analyzes published agent benchmark records. It uses no human subjects or private data. Reducing evaluation cost can encourage under-testing, so we report score errors, observed reliability, and transfer failures alongside run savings.

AI use statement

In this work, we used generative AI tools for implementing methods and the interfaces of the baselines, for providing feedback on the experiments and the pre-registration log, for cleaning and reformatting the benchmark records into response matrices, and for interpreting results. We have not used generative AI tools for generating synthetic data sets, for developing the theoretical model or conceptual framework, or for formulating or proving mathematical claims (the conformal statement of Appendix C.1.3 restates a published result), and translation and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools for creating and editing software code, creating tables and figures from the result files, drafting parts of the paper, identifying and summarizing related literature, formatting references, and checking the manuscript for inconsistencies between claims, tables and result files. We have reviewed all AI-assisted work. Every number in the paper is read from result files produced by the released code; every change to the method was pre-registered with its adoption criterion and decided by the authors on the recorded outcome (Appendix B.6); AI-generated code was reviewed and its outputs checked against the baselines' released implementations by the authors; and AI-suggested related work was verified against the cited papers. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

References

Anastasios N. Angelopoulos, Stephen Bates, Clara Fannjiang, Michael I. Jordan, and Tijana Zrnic. Prediction-powered inference. *Science*, 382(6671):669–674, 2023a.

Anastasios N. Angelopoulos, John C. Duchi, and Tijana Zrnic. PPI++: Efficient prediction-powered inference. *arXiv preprint arXiv:2311.01453*, 2023b.

Anastasios N. Angelopoulos, Stephen Bates, Adam Fisch, Lihua Lei, and Tal Schuster. Conformal risk control. In *International Conference on Learning Representations*, 2024.

Ofir Arviv, Kristjan Greenewald, Yotam Perlitz, Hadar Mulian, Michal Shmueli-Scheuer, and Leshem Choshen. Stop guessing when to stop testing: Efficient model evaluation with just enough data. *arXiv preprint arXiv:2607.08522*, 2026.

Esma Balkır et al. Confident rankings with fewer items: Adaptive LLM evaluation with continuous monitoring. *arXiv preprint arXiv:2601.13885*, 2026.

Varun Pratap Bhardwaj. Agentassay: Token-efficient regression testing for non-deterministic ai agent workflows. *arXiv preprint arXiv:2603.02601*, 2026.

Eric T. Bradlow, Howard Wainer, and Xiaohui Wang. A Bayesian random effects model for testlets. *Psychometrika*, 64(2):153–168, 1999. doi: 10.1007/BF02294533.

Robert L. Brennan. *Generalizability Theory*. Springer, 2001.

Lee J. Cronbach, Goldine C. Gleser, Harinder Nanda, and Nageswari Rajaratnam. *The Dependability of Behavioral Measurements: Theory of Generalizability for Scores and Profiles*. Wiley, 1972.

Paul De Boeck. Random item IRT models. *Psychometrika*, 73(4):533–559, 2008. doi: 10.1007/s11336-008-9092-x.

Kefeng Duan, Dewu Zheng, Yanlin Wang, Xiwen Wang, Ensheng Shi, Xilin Liu, Yuchi Ma, Jiachi Chen, Mingwei Liu, and Zibin Zheng. Efficient SWE agent benchmarking via trajectory-aware evaluation. *arXiv preprint arXiv:2609.01603*, 2026.

Susan E. Embretson and Steven P. Reise. *Item Response Theory for Psychologists*. Lawrence Erlbaum Associates, Mahwah, NJ, 2000.

Chris Ge, Daria Kryvosheieva, Daniel Fried, Uzay Girit, and Kaivalya Hariharan. Agent psychometrics: Task-level performance prediction in agentic coding benchmarks. *arXiv preprint arXiv:2604.00594*, 2026.

Eliya Habba, Itay Itzhak, Asaf Yehudai, Yotam Perlitz, Elron Bandel, Michal Shmueli-Scheuer, Leshem Choshen, and Gabriel Stanovsky. Growing pains: Extensible and efficient LLM benchmarking via fixed parameter calibration. *arXiv preprint arXiv:2604.12843*, 2026.

Valentin Hofmann, David Heineman, Ian Magnusson, Kyle Lo, Jesse Dodge, Maarten Sap, Pang Wei Koh, Chun Wang, Hannaneh Hajishirzi, and Noah A. Smith. Fluid language model benchmarking. *arXiv preprint arXiv:2509.11106*, 2025. Conference on Language Modeling.

Chia-Yu Hsu and Shubhanshu Shekhar. Efficient sequential evaluation of large language models. *arXiv preprint arXiv:2607.17409*, 2026.

Metod Jazbec, Alexander Timans, Tin Hadži Veljković, Kaspar Sakmann, Dan Zhang, Christian A. Naesseth, and Eric Nalisnick. Fast yet safe: Early-exiting with risk control. In *Advances in Neural Information Processing Systems*, 2024.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? *arXiv preprint arXiv:2310.06770*, 2023.

Sayash Kapoor et al. Holistic agent leaderboard: The missing infrastructure for AI agent evaluation. *arXiv preprint arXiv:2510.11977*, 2025. International Conference on Learning Representations.

Bora Kargi and David Salinas. From uncertain judgments to calibrated rankings: Conformal Elo estimation for LLM evaluation. *arXiv preprint arXiv:2606.13221*, 2026.

Alex Kipnis, Konstantinos Voudouris, Luca M. Schulze Buschoff, and Eric Schulz. metabench: A sparse benchmark of reasoning and knowledge in large language models. *arXiv preprint arXiv:2407.12844*, 2024.

Anany Kotawala. Resolution diagnostics for paired LLM evaluation. *arXiv preprint arXiv:2605.30315*, 2026.

Peiyu Li, Xiuxiu Tang, Si Chen, Ying Cheng, Ronald Metoyer, Ting Hua, and Nitesh V. Chawla. Adaptive testing for LLM evaluation: A psychometric alternative to static benchmarks. *arXiv preprint arXiv:2511.04689*, 2025.

John Michael Linacre. *Many-Facet Rasch Measurement*. MESA Press, 1989.

Frederic M. Lord. *Applications of Item Response Theory to Practical Testing Problems*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1980.

Frederic M. Lord and Melvin R. Novick. *Statistical Theories of Mental Test Scores*. Addison-Wesley, 1968.

Evan Miller. Adding error bars to evals: A statistical approach to language model evaluations. *arXiv preprint arXiv:2411.00640*, 2024.

Zairah Mustahsan, Abel Lim, Megna Anand, Saahil Jain, and Bryan McCann. Stochasticity in agentic evaluations: Quantifying inconsistency with intraclass correlation. *arXiv preprint arXiv:2512.06710*, 2025.

Franck Ndzomga. Efficient benchmarking of AI agents. *arXiv preprint arXiv:2603.23749*, 2026.

Bitya Neuhof et al. Quantifying ranking uncertainty in LLM benchmarks. *arXiv preprint arXiv:2607.16259*, 2026a.

Bitya Neuhof et al. Rank intervals for leaderboards: A hierarchical framework for model evaluation. *arXiv preprint arXiv:2606.08679*, 2026b.

Toby D. Pilditch. Knowing when to stop: Bayesian optimal stopping for LLM evaluations. *arXiv preprint arXiv:2608.14425*, 2026.

Felipe Maia Polo, Lucas Weber, Leshem Choshen, Yuekai Sun, Gongjun Xu, and Mikhail Yurochkin. tinybenchmarks: evaluating llms with fewer examples. *arXiv preprint arXiv:2402.14992*, 2024.

Liran Ringel, Regev Cohen, Daniel Freedman, Michael Elad, and Yaniv Romano. Early time classification with accumulated accuracy gap control. In *International Conference on Machine Learning*, 2024.

Zachary S. Siegel, Sayash Kapoor, Nitya Nadgir, Benedikt Stroebl, and Arvind Narayanan. CORE-bench: Fostering the credibility of published research through a computational reproducibility agent benchmark. *arXiv preprint arXiv:2409.11363*, 2024.

The OpenHands Team. The OpenHands index. https://index.openhands.dev, 2026. Continually updated leaderboard reporting macro-average score, cost per instance and runtime per instance across five categories of software engineering work; the Issue Resolution category is measured on SWE-bench Verified. Accessed 14 September 2026.

The Terminal-Bench Team. Terminal-bench: A benchmark for AI agents in terminal environments. https://www.tbench.ai, 2025. Leaderboard submissions must run at least five trials of every task; see https://github.com/harbor-framework/terminal-bench.

Sang Truong, Yuheng Tu, Percy Liang, Bo Li, and Sanmi Koyejo. Reliable and efficient amortized model-based evaluation. *arXiv preprint arXiv:2503.13335*, 2025.

Wim J. van der Linden and Cees A. W. Glas. Capitalization on item calibration error in adaptive testing. *Applied Measurement in Education*, 13(1):35–53, 2000.

Howard Wainer, Neil J. Dorans, Ronald Flaugher, Bert F. Green, and Robert J. Mislevy. *Computerized Adaptive Testing: A Primer*. Lawrence Erlbaum Associates, 2 edition, 2000.

Ganghua Wang et al. Cer-Eval: Certifiable and cost-efficient evaluation framework for LLMs. *arXiv preprint arXiv:2505.03814*, 2025a.

Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Openhands: An open platform for ai software developers as generalist agents. In *International Conference on Learning Representations*, volume 2025, pp. 65882–65919, 2025b.

Ian Waudby-Smith and Aaditya Ramdas. Estimating means of bounded random variables by betting. *Journal of the Royal Statistical Society Series B*, 86(1):1–27, 2024.

Skyler Wu, Yash Nair, and Emmanuel J. Candès. Efficient evaluation of LLM performance with statistical guarantees. *arXiv preprint arXiv:2601.20251*, 2026.

Yuexin Wu and Vasile Rus. Safe to stop? risk-constrained stopping for sequential clinical diagnosis agents. *arXiv preprint arXiv:2609.09678*, 2026.

John Yang, Carlos Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Zhijian Zhou, Zesheng Ye, Zhaorun Chen, Bo Li, and Feng Liu. CELEUS: Certifiable and efficient LLM evaluation via E-processes. *arXiv preprint arXiv:2606.20820*, 2026.

Algorithm 1 The per-agent protocol: evaluating one agent to a score tolerance ε

Require: training agents' records; tolerance ε; reliability level $1-\alpha$; task cap $K_{\max} = \min(N, 200)$

1: Fit the selection and scoring models (Section 3.2); learn the calibration factor $c_{1-\alpha}(\cdot)$ and the stopping bias by replay on the training agents (Section 3.4)
2: Initialize the posterior over θ_i from the training prior; $Q_i \leftarrow \emptyset$; $K \leftarrow 0$
3: **repeat**
4: $\quad j \leftarrow \arg\max_{j \notin Q_i} I_j(\hat{\theta}_i)$ under the selection model ▷ or a rerun, Section 3.5
5: $\quad$ Run task j, observe Y_{ij1}; $Q_i \leftarrow Q_i \cup \{j\}$; $K \leftarrow K+1$; update the posterior over θ_i
6: $\quad$ Compute $\hat{s}_i$ (Equation 3) and the predicted error $u_i(K)$ under the scoring model
7: **until** $c_{1-\alpha}\big(u_i(K)\big)\, u_i(K) \le \varepsilon$ or $K = K_{\max}$
8: **return** $\hat{s}_i$ minus the stopping bias, and K

A The Model and the Budget Rules

A.1 Response model and implementation details

The repeated-attempt fit integrates the cell interaction rather than fitting each observed run separately. For the model in Equation 5, we estimate abilities, difficulties, discriminations, and interaction scales by maximum a posteriori estimation. The cell likelihood uses 21-point Gauss–Hermite quadrature over ε_{ij}. The priors are $\theta_i \sim \mathcal{N}(0,1)$, $\log a_j \sim \mathcal{N}(0, 0.7^2)$, $b_j \sim \mathcal{N}(0, 3^2)$, and a half-normal prior of scale 3 for σ_j. For a test agent, ability is integrated on a grid. The observed attempts update the posterior over the cell interaction, which predicts subsequent attempts.

The single-attempt fit uses a separate regularized 2PL implementation. It alternates Newton updates of abilities, difficulties, and discriminations for 60 iterations. A penalty of weight λ pulls discrimination toward 1, and values are clipped to $[0.4, 2.5]$. The grid for test-agent ability contains 161 points in $[-4, 4]$. The normal prior uses the training agents' abilities' mean and standard deviation. Section D.13 describes internal selection of λ and the two fits used for administration and prediction.

The misfit check uses a reference built from training agents only. After one run on each task, a test agent receives the score

$$\ell_i = \frac{1}{N} \log \int \prod_j P(y_{ij} \mid \theta)\, \pi(\theta)\, d\theta.$$

We split training agents once, fit on one part, and compute the same score on the other part using one sampled attempt per task. A test-agent score below the fifth percentile triggers the fallback. Half of every later batch then follows a uniform order. Both the percentile and the batch fraction were fixed for the experiments in Table C.18.

A.2 The 2PL model and maximum-information testing

FLUID and our method use the two-parameter logistic (2PL) model. ATLAS defaults to a three-parameter model with a guessing term. Each agent has ability θ_i. Each task has discrimination $a_j > 0$ and difficulty b_j. The 2PL model gives

$$p_{ij} = \Pr(Y_{ijr} = 1 \mid \theta_i) = \sigma(a_j(\theta_i - b_j)), \qquad \sigma(x) = \frac{1}{1+e^{-x}}. \tag{9}$$

Item parameters are fitted on training agents. For a test agent they stay fixed, and only θ_i is updated. The Fisher information of task j at ability θ is

$$I_j(\theta) = a_j^2 p_j(\theta)\big(1 - p_j(\theta)\big). \tag{10}$$

Maximum-information testing runs the unused task with the largest $I_j(\hat{\theta}_i)$ and then updates $\hat{\theta}_i$. After a task set $Q_{i,K}$, the standard error of the ability estimate is

$$\mathrm{SE}_i(K) = \Big(\sum_{j \in Q_{i,K}} I_j(\hat{\theta}_i) \Big)^{-1/2}. \tag{11}$$

How the baselines fix the budget. FLUID's released code administers a fixed number of items $n_{\max}$ chosen by the user (Hofmann et al., 2025). Its paper also discusses a dynamic stopping criterion. The released code does not implement it, and we do not evaluate it. ATLAS stops when $\mathrm{SE}_i(K) \le \epsilon$ after at least 30 items, with $\epsilon \in \{0.1, 0.2, 0.3\}$ in its paper (Li et al., 2025). The threshold controls ability precision, not ranking quality. Random subsets use a budget given by the user.

How the baselines treat repeated attempts. The released adaptive IRT pipelines take one binary response per cell. The random-subset estimator can also accept a pass rate. The binomial IRT of Ge et al. (2026) accepts several attempts per cell but treats them as independent draws with the same success probability. Section 3.5 shows that this assumption is wrong for agents.

Calibration of the stopping rule. The factor $c_{1-\alpha}(u)$ and the stopping bias of Section 3.4 are learned from ten internal splits of the training pool. Each split fits the model on three quarters of the training agents with the split's λ and replays Algorithm 1, without its stopping test, on the remaining quarter. A split whose fitting part has fewer than eight agents is skipped (Appendix C.1.6). Every replayed agent contributes one pair (u, e) per budget from $K = 4$ to $K_{\max}$, and the pairs of all ten splits are pooled. The bin edges are the deciles of the pooled u, so every bin holds a tenth of the pairs. Within a bin the factor is the $(1-\alpha)$-quantile of e/u. A bin with fewer than 20 pairs takes the quantile of all pairs instead. The factor is a step function of u and is not smoothed or forced to be monotone, and a test agent's u outside the pooled range falls into the first or last bin. The stopping bias is read from the same replays. Each internal agent is stopped by the calibrated rule, and its signed error at that count is recorded. These errors are grouped into terciles of the stopping count, and the mean of the tercile that contains a test agent's own stopping count is subtracted from its score. A tercile with fewer than ten agents uses the pooled mean. If every internal split is skipped, no factor is learned and the rule stops on u alone.

Implementation of our procedure. The 2PL is fitted by alternating maximum a posteriori updates for abilities and task parameters with the quadratic penalty $\lambda \sum_j (a_j - 1)^2$ and a cap $a_j \in [0.4, 2.5]$; λ is chosen from $\{1, 3, 10, 30, 100\}$ by internal validation on the training agents (Appendix D.13). A test agent's ability is tracked on a 161-point grid over $[-4, 4]$ with a normal prior fitted to the training agents' abilities; selection evaluates the information at the posterior mean. Reruns are bought in batches of eight cells. The per-agent rule runs to at most $K_{\max} = \min(N, 200)$ tasks: on SWE-bench Verified and Lite an agent that has not stopped by then is returned at 200 with its current estimate, and Table 2 counts it at the cap. No task exposure cap is applied: capping exposure lowered τ on these banks. The score of Equation 3 plugs in the posterior mean; the task-level probability experiments instead average $p_j(\theta)$ over the posterior. The certificate and the fixed-budget ranking of Table C.23 rank agents by the posterior mean ability under the selection fit; Tables C.25, C.26 and 4 rank them by the predicted full-bank score of Equation 3 (Table B.2).

A.3 CONSTRUCTION OF THE BUDGET CERTIFICATE

From estimation error to rank correlation. Let V be the variance of training agents' abilities. Let $v(K)$ be the error variance after K adaptively selected tasks. We estimate it by replaying the adaptive procedure on the training agents and averaging the inverse accumulated Fisher information:

$$v(K) = \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \Big(\sum_{j \in Q_{i,K}} I_j(\hat{\theta}_{i,t(j)-1}) \Big)^{-1}, \tag{12}$$

where $\mathcal{C}$ is the training set and $t(j)$ is the step at which task j is run. Independent measurement error attenuates the correlation between estimated and true ability to

$$r(K) = \sqrt{\frac{V}{V + v(K)}}. \tag{13}$$

For a bivariate normal pair, a Pearson correlation r implies a Kendall correlation

$$\tau_{\mathrm{G}}(K) = \frac{2}{\pi} \arcsin r(K). \tag{14}$$

Abilities are not exactly normal, and a finite bank adds error that Fisher information ignores. We correct with one offset d, measured on the replay on the training agents at the full bank. The point prediction is

$$\widehat{\tau}(K) = \tau_{\mathrm{G}}(K) + d. \tag{15}$$

The mapping from K to $\widehat{\tau}(K)$ is therefore a Gaussian approximation corrected by one offset learned by replay, and the margin $m_q(K)$ below absorbs what the offset misses on the internal splits. We did not test a nonparametric mapping, for example one read directly off the replayed Kendall correlations at each K, nor an offset that varies with K.

From a prediction to a certificate. A point prediction does not state how often the target is missed. We split the training agents 30 times into an internal fitting set and an internal test set of the same size as the new cohort. For every K, we record how much the certificate fitted on the internal fitting set overpredicts the Kendall correlation measured on the internal test set. For a stated reliability q, let $m_q(K)$ be the q-quantile of this overprediction. The certificate chooses

$$K^* = \min\bigl\{K \geq 4 :\ \widehat{\tau}(K) - m_q(K) \geq \tau^*\bigr\}. \tag{16}$$

If no budget qualifies, it returns the whole bank and states that the target is not certified. With 30 internal splits, high quantiles are coarse, so the stated reliability is an estimate, not a guarantee. Section C.1.8 checks it on held-out splits. A second variant multiplies the point-prediction budget by a factor c, chosen as the smallest factor that hits the target on every internal split.

A.4 Certificate derivation: attenuation and variance components

A.4.1 Attenuation from score noise

Write the estimated score at budget K as $\hat{s}_{i,K} = s_i + e_{i,K}$, where s_i is full-bank performance and $e_{i,K}$ is measurement error. Under mean-zero error independent of s_i,

$$\mathrm{Corr}(s_i, \hat{s}_{i,K}) = \frac{\mathrm{Var}(s_i)}{\sqrt{\mathrm{Var}(s_i)\mathrm{Var}(s_i + e_{i,K})}} = \sqrt{\frac{V}{V + v(K)}}. \tag{17}$$

The Gaussian copula identity maps this Pearson correlation to Kendall correlation through Equation 14. The identity is an approximation for agent cohorts. Adaptive replay provides the full-bank offset d in Equation 15.

For a desired stated reliability q, internal agent splits produce errors

$$e^{\tau}_{h,K} = \widehat{\tau}_h(K) - \tau_h(K), \tag{18}$$

where h indexes the internal split. The margin m_q is the empirical q-quantile of overprediction error. Subtracting it from the point prediction yields Equation 16. This construction is deliberately one-sided because underpredicting ranking quality wastes tasks, while overpredicting it misses the user's target.

A.4.2 Finite banks and repeat noise

Let S_i be a subset score and θ_i its whole-bank expectation. Task and run noise decompose as

$$\mathrm{Var}(S_i - \theta_i) = A_i \left(\frac{1}{K} - \frac{1}{N}\right) + \frac{B_i}{KR}, \tag{19}$$

where A_i is between-task variation and B_i is within-task run variation. Sampling K tasks without replacement gives the finite-bank factor; averaging R attempts gives the run-noise term. Repeated training cells identify both components. Single-attempt benchmarks identify only their combined effect without further assumptions.

A.5 Item calibration from a single attempt

Terminal-Bench's repeated attempts also let us test whether a single recorded run is sufficient to calibrate items. Discrimination estimated from a single attempt correlates 0.891 with the all-attempt estimate and is biased downward, 1.407 versus 1.527. Difficulty correlation is 0.982. Despite this parameter noise, the largest Kendall difference at $K = 5, 10, 20, 40$ is 0.007, and no interval excludes zero.

A.6 The design space

Every method in this family, ours included, is an assembly of seven choices. FLUID, ATLAS and tinyBenchmarks differ from one another by the options they take, not by their overall shape. We name the choices here, report in Section 6 which of them change the answer on agentic benchmarks, and state the assembly that survives in Section 3.

C1. How item parameters are estimated. Joint maximum likelihood, marginal maximum likelihood, or a hierarchical Bayesian fit. FLUID uses a Bayesian 2PL with hierarchical priors; ATLAS uses marginal maximum likelihood for a 3PL.

C2. How strongly discriminations are shrunk. Not at all, by a fixed amount, or by an amount chosen from the training pool itself. Released calibrations of the baselines apply no explicit shrinkage.

C3. How the next task is selected. Maximum Fisher information at the current ability estimate, information integrated over the current posterior, a shared set of anchor tasks followed by adaptive selection, or one designed panel used for every agent.

C4. How ability is estimated. A maximum a posteriori point estimate, as in FLUID, or the mean of a posterior on a grid.

C5. Whether task exposure is limited. Adaptive tests for people cap how often an item may be administered, to protect the item pool. TITRATE applies no such cap; without it the ranking is no worse at any budget (Appendix B.6).

C6. How the budget is decided. A fixed share of the bank, a count fixed in advance for a stated target, or a per-agent stopping rule that runs until an estimate is precise enough. ATLAS is the only baseline whose released code stops on its own, on the standard error of ability.

C7. How repeated attempts are treated. As independent draws, reduced to a majority label, or with an agent–task interaction that separates a poor match from a bad run. If repeats are modelled, a further choice is how to allocate them: uniformly, by a fixed schedule per task, or per agent by expected variance reduction.

Sections 3.5 to 3.5 give the formal content of the options we end up using: the response model for repeated attempts (C7), the adaptive test (C3, C4), the budget certificate (C6) and the rerun decision (C7). Section 6 then measures every choice on the same splits.

B Experimental Protocol

This appendix records how every number in the paper was produced: the benchmarks and splits, the released code each baseline was run with, the data construction, the compute, and the criteria we fixed before running.

B.1 Benchmarks, splits and experiments

B.1.1 Benchmarks and data sources

We study five public benchmarks spanning terminal work, code repair, scientific reproduction, and customer-service tool use. Terminal-Bench 2.0 has 3–10 attempts per cell, with median 5 (Ge et al., 2026). SWE-bench uses public benchmark results (Jimenez et al., 2023); CORE-bench Hard and τ-bench airline use the Holistic Agent Leaderboard (Kapoor et al., 2025; Siegel et al., 2024; Yao et al., 2024). Table 1 in the main text gives the size of each pool and bank.

B.1.2 Agent-level split protocol

We split agents, not tasks. Each split draws 75% of a benchmark's agents as training agents and holds out 25% as test agents. This gives training/test counts of 95/32 for Terminal-Bench 2.0, 130/44

Method	Models used to calibrate items	Adaptive selection	Models repeats	Allocates repeats	Stops on
tinyBenchmarks (Polo et al., 2024)	395	no (fixed anchors)	no	no	—
metabench (Kipnis et al., 2024)	>5,000	no (fixed subset)	no	no	—
Growing Pains (Habba et al., 2026)	395–428	no (fixed anchors)	no	no	—
Mid-range panel (Ndzomga, 2026)	leaderboard history	no (fixed panel)	no	no	—
PTA-IRT (Duan et al., 2026)	14–86 agents	no (fixed subset)	no	no	—
FLUID (Hofmann et al., 2025)	102	yes	no	no	ability SE vs. rank gap
ATLAS (Li et al., 2025)	3,467–4,680	yes	no	no	ability SE $\leq \epsilon$, ≥ 30 items
Ours	**10–130 agents**	**yes**	**yes**	**yes**	**calibrated score error** $\leq \varepsilon$

Table B.1: **What each method assumes and decides.** "Models repeats": the response model treats attempts on one task as dependent; "allocates repeats": the procedure chooses which cells to run again. Last column: the quantity the stopping rule is expressed in. FLUID and ATLAS stop on the standard error of a latent ability, not on the error of the reported score.

for SWE-bench Verified, 63/21 for SWE-bench Lite, 49/17 for CORE-bench Hard, and 35/12 for τ-bench airline. We use 30 random splits per benchmark. Every method sees the same training agents and the same test agents in every split.

What each method may use. Training agents reveal their full protocol. Every method fits its model on them, and our certificate also computes its margin on them. A test agent reveals only the runs a method buys. No method sees any response of a test agent before it fixes its budget.

Reference. The target is the recorded official score of Equation 2: the full-bank pass rate, averaged over attempts on Terminal-Bench 2.0. For the single-attempt comparisons, every method receives the majority label of a Terminal-Bench cell, because the released code of every baseline accepts only one binary response per cell. The rerun experiments of Section C.2.2 instead draw individual attempts. There, a run returns one archived attempt of the cell, drawn without replacement, so that five runs reproduce the official protocol exactly.

Redrawing the splits. Every number we report comes from one family of 30 splits, generated from fixed seeds. We drew a second family from different seeds and recomputed our score curve and our task budgets on all five benchmarks, with the shrinkage chosen inside each pool so that nothing carries over from the first family. The task budgets are stable: reaching four, three and two points costs within 1% to 12% of what it costs on the first family, and every target that is out of reach on one family is out of reach on the other. The second family's mean falls inside the first family's 95% interval in 20 of the 25 budget cells. Five misses is what independent redrawing predicts rather than a sign of instability: two independent family means differ with $\sqrt{2}$ times the standard error of one, so 4.1 of 25 are expected to fall outside. We checked this directly with five independent families of 30 splits. The standard deviation of the five family means, divided by the bootstrap standard error inside a family, has a median of 1.03 across the fifteen cells, and the same ratio has a median of 0.93 when the splits carry no family effect at all. The intervals we report therefore already carry the variation that comes from drawing a different set of training agents, because the bootstrap resamples splits and not agents within one split. We did not rerun the baselines on the further families, because that means rerunning their released code, so this measurement bounds the stability of our own numbers and not of the comparisons.

B.1.3 EXPERIMENTS

The results come in two parts, one per kind of benchmark.

Benchmarks with one attempt per task (SWE-bench Verified, SWE-bench Lite, CORE-bench Hard, τ-bench airline).

1. **Score prediction**, the primary metric. After 10% or 20% of the bank, each method that outputs a score predicts the full-bank pass rate. We report the mean absolute error. Budgets below four tasks, the smallest budget our certificate selects, are reported only in the appendix; this rule applies to every method.

2. **Predicting unrun tasks.** Each method predicts success on every task the test agent did not run.
3. **Ranking.** Kendall τ at 10% of the bank, the metric of the FLUID and ATLAS papers; the appendix adds 20%, 30% and 50%.
4. **The same ranking quality with fewer tasks.** The certificate fixes, from training agents only, the budget our method needs to reach each baseline's mean τ at 10% or 20%.
5. **Calibration-pool size.** We shrink the training agents to m and find the smallest m within 0.5 percentage points of the full pool.
6. **A time split.** Earlier agents calibrate and later agents are new, on the benchmarks with submission dates.
7. **A budget fixed in advance.** We compare ATLAS's stopping rule with the certificate and check the certificate's stated reliability.

A benchmark with repeated attempts (Terminal-Bench 2.0). Cost is counted in runs, and a run returns one archived attempt of the cell, drawn without replacement.

8. **Runs needed against every baseline.** Each baseline runs its selected tasks once, or five times as the protocol requires. Our method runs tasks once and decides reruns.
9. **Predicting reruns.** A test agent runs each task once, and the model predicts the remaining archived attempts of each cell.
10. **Deciding reruns.** All 89 tasks are run once, and further runs go wherever a method chooses. The criteria were fixed before the experiment was run (Appendix B.5).

Where ATLAS could not be run. ATLAS's released pipeline fits a 3PL model with `mirt`. On the smallest pools it fails outright: on Terminal-Bench 1.0, with 15 training agents and 80 tasks, every one of the six splits we attempted stopped with `Could not invert orthogonal complement matrix`, and on Terminal-Bench 4.0 two of fifteen attempted splits stopped the same way. It also failed on 2 of 30 splits of τ-bench airline, which we report in the main tables. We therefore do not report ATLAS on Terminal-Bench 1.0 and report it on the other small releases only where it ran. This is a property of the released code at this pool size, not a tuning choice of ours: we made no change to its fitting routine; the documented chunking and linking boundary fix are listed in Appendix B.2.

Benchmarks held out for confirmation. Our five main benchmarks are also the ones we designed on, and the three further benchmarks we set aside are a one-time confirmation for the decisions that were frozen before they were touched, not an untouched set for the final system. The development log records the order: the misfit check and the exposure decision were confirmed on them prospectively, while the shrinkage rule was revised after a calibration failure on a further Terminal-Bench release used only in development, so for that rule these releases are development data. Appendix B.6 gives the chronology. The three additional releases are: SWE-bench Full (24 agents, 2294 tasks), Terminal-Bench 1.0 (20 agents, 80 tasks) and Terminal-Bench 4.0 (18 agents, 66 tasks). One of the three carries repeated attempts, which is what lets us confirm the rerun decisions. Every adoption reported below states the criterion, then the result on the benchmarks used for design, then the result on these four.

What was fixed before running. The development log records adoption criteria for the experiments below; some broader design choices followed results seen during development, recorded in the log released with the code, and the criterion was not revised afterwards. When a criterion turned out to be the wrong one, as happened once, we say so and report the result under both readings rather than silently switching. Of the 34 entries, six changed the method, 18 were rejected by their own criterion and 10 were descriptive or reported with limits; Appendix B.6 lists all of them with their criteria and outcomes.

B.2 BASELINES: AUTHORS' RELEASED CODE

We designate one family of primary comparisons and correct it for multiplicity. The family is the score error against ATLAS's estimator and against random subsets, on the four single-attempt

benchmarks at every budget of at least four tasks: 36 paired comparisons. We report Holm-corrected results for that family and unadjusted 95% bootstrap intervals everywhere else, labelled as such. A non-significant difference is never read as equivalence.

Agents on one benchmark are not independent: several versions often share a scaffold, and several scaffolds often share a model. We therefore repeat the primary comparisons with a bootstrap that resamples scaffold families rather than splits. The pools contain 68, 48, 13, 14 and 40 families, with the largest family holding 10%, 8%, 41%, 28% and 25% of the agents. Under that resampling our advantage over random subsets stays significant in all ten cells, by 1.60 to 8.66 percentage points, and our advantage over ATLAS stays significant where it was significant before, on SWE-bench Lite at both budgets and on CORE-bench Hard at four tasks; the remaining cells against ATLAS were ties under either resampling. The family intervals are wider, which is the part that treating same-family agents as independent hides.

Every baseline is run from the authors' released implementation. Our code only converts data formats and records results; the changes we had to make are listed exhaustively.

- **Random** and **Random + IRT** are the two baselines shipped with FLUID's code (`random_accuracy`, `random_ability`): a shared random subset scored by pass rate, or by the 2PL ability estimate. Knob: subset size.
- **FLUID** (Hofmann et al., 2025) uses `fluid_benchmarking` from `allenai/fluid-benchmarking` (commit `d704b14`): item parameters fitted with its `irt/fit_irt_model.py` (py-irt, 2PL, hierarchical priors), maximum-information selection, MAP ability, start ability equal to the training agents' mean, as in its experiment script. Knob: $n_{\max}$.
- **ATLAS** (Li et al., 2025) runs its R pipeline from `Peiyu-Georgia-Li/ATLAS` (commit `c0bc19b`): `mirt` 3PL fitting, parameter linking, and the `catR` adaptive test (maximum information with a random choice among the five most informative items, EAP ability, stop when the standard error falls to ϵ after at least 30 items, at most 500). We add our benchmarks to its configuration tables. As in its pipeline, banks with more than 110 tasks are fitted in chunks of about 105 tasks and linked by its own script. We make two edits to its scripts, neither touching the fit, the adaptive test or the estimator. Its linking script iterates `2:length(x)`, which in R reads a nonexistent second parameter block when there is only one. Its fitting script computes the `mirt` M_2 fit statistic after fitting and before writing the item parameters; on the smallest pools M_2 fails to invert a matrix and aborts the script, so we let that diagnostic fail without aborting. We use the thresholds from its paper, $\epsilon \in \{0.1, 0.2, 0.3\}$. Tables 4 and C.23 use the ranking output of its released adaptive script. Every table that scores a full-bank score, including the ranking columns of Table C.25, uses its `04_pirt_accuracy.r`, run unmodified, with one conversion: that script reports the rate over the cleaned bank from which it has removed the tasks every training agent passed or failed, and the full-bank estimate is the cleaned-bank rate times the number of cleaned tasks, plus the number of tasks every training agent passed, divided by N. Its adaptive script skips a test agent that passes or fails everything; we record such an agent at its true pass rate, which favours ATLAS.
- **Mid-range panel** (Ndzomga, 2026): the filter keeps tasks with historical pass rates between 30% and 70% and scores an agent by its raw pass rate on them. At a fixed budget K we keep the K tasks whose training pass rate is closest to one half, shared by all test agents of a split, and equate the panel rate to the full bank by the classical panel's linear regression; the raw panel rate is a poor estimate of the full-bank score (14–18 points off at a tenth of the bank) because mid-range tasks are not the bank, so the equated score is what the tables report. Its ranking uses the raw panel rate, as in the filter.
- **PTA-IRT** (Duan et al., 2026) runs the authors' package (`pta/`) on our response matrices and our 30 splits with its frozen hyperparameters and its stratified training subset, in its no-trajectory mode, because the public leaderboards do not release trajectories for every agent. Budgets 5–90% of the bank; the table reads the budget closest to 10% and 20%.
- **tinyBenchmarks** (Polo et al., 2024) runs the authors' pipeline from `felipemaiapolo/tinyBenchmarks`: a multidimensional 2PL fitted with `py-irt` under hierarchical priors at their learning rate of 0.1, the dimension chosen from $\{5, 10\}$ by

their own validation split, anchor points from weighted k-means on the training responses, and their `irt`, `p-IRT` and `gp-IRT` estimators with λ from their `get_lambda`. One deviation, fixed in advance and applied to every benchmark: at their default of 2000 training epochs the fit diverges on our agent pools, so we also report 300 epochs, and we report the failure counts at the default.

- **Ours** uses maximum-information selection without exposure control, shrunken 2PL parameters and an EAP ability with a benchmark prior (Section 3.2). Its budget is either given, as for the baselines, or fixed by the certificate (Equation 16).

Table B.2 lists, for each experiment, which quantity of ours and of each baseline is scored.

Experiment	Quantity of ours scored	Quantity of each baseline scored
Score and ranking at a fixed budget (Tables C.25, C.26 and C.27)	predicted full-bank score	predicted full-bank score from its released fit; ATLAS: p-IRT score estimate (`04_pirt_accuracy.r`); Random: observed pass rate on the tasks run
Tasks to a target ranking (Tables 4 and C.4)	predicted full-bank score	FLUID and ATLAS: ranking output of the released adaptive script; Random: observed pass rate on the tasks run
Operating point and stopping (Tables 3, 2, D.7 and D.8)	predicted full-bank score	predicted full-bank score from its released fit; ATLAS: p-IRT score estimate (`04_pirt_accuracy.r`); Random: observed pass rate on the tasks run
Cohort certificate and fixed-budget ranking (Table C.23)	posterior mean ability under the selection fit	FLUID and ATLAS: ranking output of the released adaptive script; Random: observed pass rate on the tasks run; Random + IRT: ranking output of FLUID's released `random_ability` script
Same guarantee under conformal risk control (Table C.3)	predicted full-bank score	predicted full-bank score from its released fit; ATLAS: p-IRT score estimate (`04_pirt_accuracy.r`)
Reruns (Tables C.19, C.16, C.17 and C.21)	predicted official score under the repeat-aware posterior	uniform reruns: observed mean of the attempts run in each cell, or our model's predicted official score; binomial 2PL: its own predicted official score; complete passes: observed pass rate on the attempts run

Table B.2: **The quantity each experiment scores.** One line per experiment. For TITRATE, the predicted full-bank score is Equation 3: observed outcomes on the tasks run plus model probabilities on the rest.

B.3 DATA CONSTRUCTION AND REPRODUCTION PROTOCOL

The unit of splitting is an agent row in the response matrix. Task columns are never held out: all training-agent columns are needed to estimate item difficulty and discrimination. Within each of the 30 outer splits, every method receives the same training rows and the same test-agent rows.

Terminal-Bench 2.0 contains 127 agents and 89 tasks, with 95 training and 32 test agents per split. Its truth averages at least 5 attempts per cell. SWE-bench Verified contains 174 agents and 500 tasks, split 130/44. SWE-bench Lite contains 84 agents and 300 tasks, split 63/21. CORE-bench Hard contains 66 agents and 45 tasks, split 49/17. τ-bench airline contains 47 agents and 50 tasks, split 35/12. The last four matrices contain one recorded attempt per cell.

Difficulty strata and 2PL parameters are recomputed from training agents within each split. For adaptive methods, each test agent starts with no observed responses and receives tasks sequentially. For the certificate, K is fixed before that sequence begins. The full test-agent row is consulted only after scoring, to compute the reference pass rate and Kendall τ_b.

B.4 COMPUTATIONAL COST

All of our computation runs on CPU, one core per split. The expensive part is calibration: fitting the 2PL with the internal validation that chooses the penalty weight, replaying the administration

on thirty internal splits of the training pool to calibrate the certificate margin, and computing the certificate itself take between 1.5 and 18 minutes per split, 88–121 seconds on τ-bench airline and CORE-bench Hard, 6–7 minutes on Terminal-Bench 2.0, 9–11 minutes on SWE-bench Lite and 18 minutes on SWE-bench Verified. The per-agent stopping rule, which replays ten internal splits, and the score prediction for all test agents of a split take seconds: on CORE-bench Hard, all thirty splits of the stopping rule run in 16 seconds and of the score curve in about half a minute. Evaluating one test agent is a posterior update on a 161-point grid per response and one argmax over the unrun tasks, negligible next to a single run of the agent. The repeated-attempt model on Terminal-Bench 2.0 is fitted by L-BFGS with 21-point quadrature in about 20 seconds per split, and allocating reruns for the 32 test agents of a split from 89 to 356 runs takes about 40 seconds.

The released baselines cost more. ATLAS's R pipeline fits a 3PL by `mirt` in chunks of about 105 tasks and then runs its adaptive test; one split takes about an hour on CORE-bench Hard and four to five hours on SWE-bench Verified and Lite on one core. FLUID's `py-irt` fit takes 40–90 seconds per split. tinyBenchmarks' fit takes one to three minutes per split at 300 epochs and diverges on most small-pool splits. PTA-IRT trains its trajectory encoder for nine budgets per fold; ten splits take about 35 minutes with many threads. None of these costs approaches the cost of the runs they save: one task run on SWE-bench Verified costs $0.22 at the median (Appendix D.9), so the training of a bank pays for itself after a few dozen runs.

B.5 PRE-REGISTRATION OF THE RERUN-ALLOCATION EXPERIMENT

We fixed the design and the success criteria of the rerun-allocation experiment before running it. The record is kept with the code. Section C.2.3 reports the outcome; all three criteria hold.

Design. Terminal-Bench 2.0, the same 30 agent splits. Training agents fit the repeat-aware model on all their attempts. Each test agent first runs all 89 tasks once. A run returns one archived attempt of the cell, drawn without replacement, and a cell with no attempts left cannot be run again. Further runs are launched in rounds of eight per agent. The totals are 134, 178, 223, 267 and 356 runs per agent; the official protocol uses 445. The runs actually spent by every method are printed and checked to be equal at every total.

Methods. Uniform allocation with the empirical cell mean; uniform allocation with the model's estimate; a disagreement-first rule that sends runs to cells that have shown both outcomes; allocation by the binomial 2PL model; and our allocation by expected variance reduction with the model's estimate.

Metrics. The primary metric is the mean absolute error of the estimated official score. Secondary metrics are Kendall τ among test agents, the ordering accuracy of pairs whose official scores differ by less than 0.03, and the coverage of our 90% interval.

Criteria. (1) At three or more of the five totals, our error is significantly below that of uniform allocation, by a paired 95% interval over the 30 splits, and it is significantly above at none. (2) Our allocation matches the error of uniform allocation with three runs per task (267 runs) with at most 214 runs, a saving of at least 20%. (3) Our allocation beats the binomial-2PL allocation. If criterion (1) fails, we report that dynamic reruns do not save runs.

B.6 PRE-REGISTRATION LOG

Table B.3 lists every change we considered, the criterion written before the run, and the outcome. Three rules for the shrinkage strength were compared by the certificate they produce at a stated reliability of 0.95 over five benchmarks and four targets: a fixed $\lambda = 10$ gives a mean realized reliability of 0.882 with a worst cell of 0.77, selection by log loss 0.848 and 0.43, and selection by the certificate's own validation 0.883 and 0.77; we adopt the last (Section D.13).

Change considered	Criterion written before the run	Outcome
Exposure cap removed	Ranking not worse at any budget	Adopted

Change considered	Criterion written before the run	Outcome
Shrinkage chosen by pool (G1)	Score error lower on a majority of cells, none worse	Adopted
Shrinkage chosen by the certificate's own validation (G3)	Certificate reliability up, no loss elsewhere	Adopted
Two shrinkage values, one per use (G4)	No main metric significantly worse, gains elsewhere	Adopted
Misfit check with uniform fallback	No budget worse within release, rescue across releases	Adopted
Per-agent conditional margin (H2)	Per-agent reliability at 0.90 in most cells	Adopted for the guarantee variant only
Chronological confirmation of E3	Hold on the four confirmation sets	Rejected
Per-agent stopping rule (H1)	Cheaper at equal quality	Rejected
Model and scaffold decomposition as a prior (I1)	Score error lower with dense labels	Rejected
Gated decomposition prior (I2)	No cell worse	Rejected
2PL against 3PL and 4PL (K1)	A better form chosen by internal validation	Rejected; 2PL retained
Cheapest among near-maximally informative tasks (J3)	No cell worse at equal dollars	Rejected as default
Score certificate in dollars (M1)	Reliability at the stated level	Rejected; not verifiable here
Budget split across agents (O1)	Four cells better, none worse	Rejected
Runs aimed at near-tie pairs (R1, R1b)	Near-tie accuracy above even splitting	Rejected
Shrinkage chosen by ranking loss (S1)	AUC no longer below FLUID	Rejected
Ability as the reported score (Q2)	More reproducible than the pass rate	Rejected
A budget below one full pass (Q3)	Beat a full pass on three of four releases	Rejected
Paired common tasks for version decisions (P1)	Fewer runs than separate adaptive tests	Rejected; stopping half kept
Variance-robust certificate (U1)	Reliability at 0.90 under time splits	Rejected; mechanism refuted
Information discount for time shift (U2)	Reliability at 0.90 under time splits	Rejected; discount is null
Prospective validation of slimming (V1)	Beat random removal across generations	Rejected; negative result
Value of the misfit detection (Z1)	Never worse than the better of the two arms	Rejected; cost now priced
Duration as the driver of repeat noise (A1)	Holds after controlling difficulty	Rejected; pass rate is the driver
Effect of pool size on tasks needed (G2)	Descriptive	Reported
Task-level diagnostics (L1)	Descriptive	Reported
Slimmed bank (N1)	Ranking preserved at half the bank	Reported with limits
Prediction scored on common tasks (S2)	No cell worse on the common set	Reported as a second scoring
Classical test theory panel (T1)	Match us in three cells	Reported as a baseline
tinyBenchmarks released pipeline (T3)	Match us in any cell	Reported as a baseline
Human effort annotations (T2)	Difficulty correlates above 0.2	Reported as external validity
Attempts per task at a fixed run budget (W1)	A consistent optimum across releases	Reported as protocol guidance
Matched disjoint reference (X1)	Tie one full pass at equal runs	Reported; claim qualified
Training pool thinned within a release (Y1)	Advantage falls with the pool	Reported; confound removed

Table B.3: The pre-registration log. Every change was written down with its adoption criterion before the experiment ran. Of 34 entries, 6 changed the method, 18 were rejected by their own criterion, and 10 were descriptive or reported with limits. The full log, with the numbers and the verdict text, is released with the code.

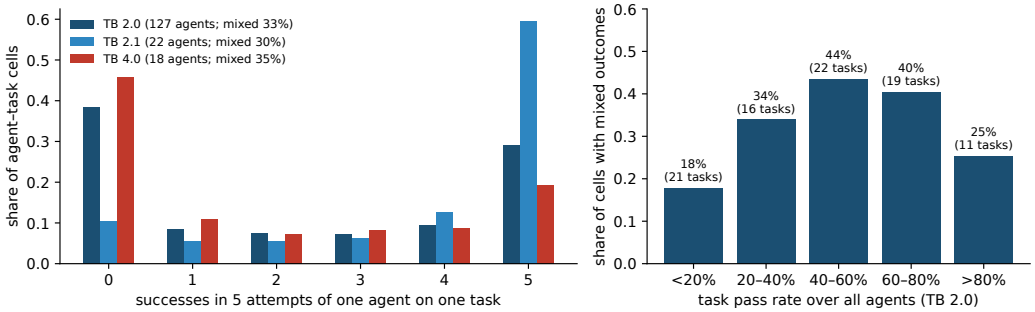

Figure C.1: Agent outcomes are stochastic. Left: number of successes in five attempts of one agent on one task, for the three Terminal-Bench releases we study. About one third of cells mix passes and failures. Right: on Terminal-Bench 2.0, mixed cells concentrate on tasks of intermediate difficulty, but every difficulty band contains them.

B.7 Comparisons that the available data limits

ATLAS on the held-out benchmarks. ATLAS's released 3PL pipeline completes on none of the three held-out sets. On Terminal-Bench 1.0 its fit stops with a linear-algebra error on all six splits, on Terminal-Bench 4.0 on two of fifteen, so the comparison cannot be completed there, and it was not run on SWE-bench Full. We record the failures.

Two comparisons the published data does not support. The score certificate is validated on the held-out benchmarks of Section D.12 but not on Terminal-Bench 2.1, whose small training pool limits internal estimation of high quantiles. The cost measurements of Section D.8 use the three Terminal-Bench releases that publish per-run cost; SWE-bench cost recovery is limited to 35 submissions from one scaffold family (Appendix D.9). These partial cost records do not validate a general budget certificate in dollars.

C Detailed Results

The results divide by benchmark family. Appendix C.1 covers the five banks with one recorded attempt per task, in the order of the main text: score prediction, per-task prediction, ranking, the tasks a target ranking costs, how many training agents are needed, the time split, and the prospective budget. Appendix C.2 covers Terminal-Bench 2.0, where runs rather than tasks are the unit and a cell can be run again. The last three parts give the tables at every budget.

C.1 Benchmarks with one attempt per task

C.1.1 Predicting each test agent's full-bank score

The most direct question about a test agent is how many tasks of the whole bank it would solve. Unlike a ranking, the answer does not depend on which other agents are evaluated with it. After running 10% or 20% of the bank, each method that outputs a score predicts the full-bank pass rate. Our estimate adds the observed outcomes of the tasks run to the model's success probabilities for the tasks not run. Table C.25 reports the mean absolute error, and Table C.27 every budget.

Our score predictions are the most accurate. With 10% of SWE-bench Verified, 50 tasks, our error is 2.34 percentage points, about 12 of the 500 tasks. ATLAS's error is 2.44 points and the random subset's 5.65. With 10% of SWE-bench Lite our error is 2.38 points, about 7 of 300 tasks. We compare at 5%, 10%, 20%, 30% and 50% of each bank, reporting only budgets of at least four tasks, a rule applied to every method (Appendix C.4 keeps the smaller ones). Against the random subset our error is significantly lower in all 18 comparisons. Against ATLAS's p-IRT estimator, the strongest baseline, it is significantly lower in 13 of 18 unadjusted and in 12 of 18 after Holm correction over the 36 primary comparisons, and never higher; the comparisons against random subsets all survive

Benchmark	Target	Random	ATLAS	TITRATE
SWE-bench Verified	error ≤ 4	$86 \pm 12^{\dagger}$	$\leq 25^{\dagger}$	$\mathbf{13 \pm 1}$
	error ≤ 3	$119 \pm 19^{\dagger}$	29 ± 6	$\mathbf{24 \pm 3}$
	error ≤ 2	$213 \pm 29^{\dagger}$	90 ± 17	$\mathbf{75 \pm 7}$
SWE-bench Lite	error ≤ 4	$76 \pm 19^{\dagger}$	$15 \pm 1^{\dagger}$	$\mathbf{10 \pm 2}$
	error ≤ 3	$117 \pm 20^{\dagger}$	$32 \pm 5^{\dagger}$	$\mathbf{20 \pm 3}$
	error ≤ 2	n.r.†	$73 \pm 8^{\dagger}$	$\mathbf{40 \pm 3}$
CORE-bench Hard	error ≤ 4	n.r.†	$16 \pm 2^{\dagger}$	$\mathbf{12 \pm 1}$
	error ≤ 3	n.r.†	n.r.†	$\mathbf{17 \pm 1}$
τ-bench airline	error ≤ 4	n.r.†	n.r.†	$\mathbf{20 \pm 2}$
TB 2.0	error ≤ 4	n.r.†	11 ± 1	$\mathbf{11 \pm 1}$
	error ≤ 3	n.r.†	19 ± 2	$\mathbf{18 \pm 2}$
	error ≤ 2	n.r.	$\mathbf{43 \pm 2}$	n.r.

Table C.1: **Tasks per test agent needed to bring the score error below 4, 3 and 2 percentage points.** Read off each method's mean curve over the 30 splits and interpolated between budgets; brackets are 95% intervals from resampling splits. Bold: fewest. n.r.: the mean curve stays above the target at every budget the method was run at, the whole bank for ours and half of it for the random subset and ATLAS. †: ours needs significantly fewer tasks. A baseline entry written $\leq K$ already meets the target at the smallest budget its released code exposes and carries no interval. Rows on which no method reaches the target are omitted (CORE-bench Hard at 2 points; τ-bench airline at 3 points; τ-bench airline at 2 points). On Terminal-Bench 2.0 every method, ours included, consumes the five recorded attempts of an administered task through its majority label, so the comparison in tasks is matched.

the correction. The one comparison the correction removes is CORE-bench Hard at four tasks, where the unadjusted p is 0.029. Four of the five ties are SWE-bench Verified, where our 2.34 points at 10% is numerically below ATLAS's 2.44, and the fifth is CORE-bench Hard with nine tasks.

Score-error targets provide a second comparison across budgets. A fixed share of the bank is not the only way to compare. Table C.1 asks how many tasks each method needs before the mean error of the predicted full-bank pass rate falls below 4, 3 and 2 percentage points. We read the answer off each method's mean curve over the same 30 splits and interpolate between budgets, the rule we also use for ranking in Section C.1.4. Our method needs the fewest tasks in 11 of the 15 cells, significantly fewer than a baseline in 18 of the 30 comparisons, and never significantly more. On SWE-bench Lite it reaches 3 points with 20 tasks, against 32 for ATLAS and 117 for the random subset, and 2 points with 40 against 73 and never. On CORE-bench Hard and τ-bench airline the random subset never reaches 4 points within the bank, and ATLAS never reaches 3 points on CORE-bench Hard.

Two entries need reading with care, and one of them is a loss. ATLAS's SWE-bench Verified cell at 4 points is censored at the smallest budget its code exposes, so the exact saving there is not identifiable from the displayed threshold; our curve is read at every integer budget and the baselines only at their five operating points. On Terminal-Bench 2.0 at the two-point target ATLAS crosses at 43 tasks and our mean curve does not cross within half the bank, 2.03 points at 44 tasks against its 1.96. Both methods consume the five recorded attempts of an administered task there, through the majority label, so the comparison is matched and the cell is a genuine loss for the task-counting pipeline. It is also the only benchmark where our procedure would not be deployed in that form: Section C.2.1 spends the same attempts individually and reaches 0.76 points at 223 runs, against 1.96 for ATLAS at the 220 runs its 44 tasks consume.

C.1.2 PREDICTING THE TASKS A TEST AGENT HAS NOT RUN

After a test agent runs 10% or 20% of the bank, the fitted model predicts whether it solves each remaining task. Table C.2 scores these predictions on the tasks it did not run. All methods see the training agents and the test agent's own selected tasks.

Benchmark	Tasks run	Tasks the method did not run: accuracy / AUC / Brier				Tasks neither ran: ours / FLUID	
		Ours	FLUID	Task pass rate	Agent pass rate	AUC	Brier
Verified	50	0.888 / 0.953 / 0.083	0.884† / 0.951† / 0.085†	0.810† / 0.889† / 0.133†	0.662† / 0.707† / 0.220†	0.955 / 0.954†	0.081 / 0.082†
	100	0.909 / 0.964 / 0.070	0.901† / 0.961† / 0.074†	0.849† / 0.920† / 0.114†	0.673† / 0.726† / 0.217†	0.965 / 0.965†	0.068 / 0.068†
Lite	30	0.885 / 0.943 / 0.085	0.878† / 0.940† / 0.090†	0.846† / 0.912† / 0.111†	0.614† / 0.651† / 0.232†	0.947 / 0.945†	0.082 / 0.083†
	60	0.915 / 0.956 / 0.069	0.897† / 0.951† / 0.077†	0.886† / 0.936† / 0.090†	0.603† / 0.672† / 0.230†	0.962 / 0.961†	0.062 / 0.063†
CORE	4	0.825 / 0.856 / 0.123	0.796† / 0.852† / 0.136†	0.795† / 0.815† / 0.140†	0.688† / 0.659† / 0.241†	0.862 / 0.856†	0.121 / 0.128†
	9	0.859 / 0.873 / 0.105	0.828† / 0.879* / 0.119†	0.837† / 0.834† / 0.121†	0.628† / 0.688† / 0.223†	0.888 / 0.889	0.101 / 0.103†
τ-bench	5	0.713 / 0.783 / 0.187	0.707† / 0.777† / 0.189	0.698† / 0.768† / 0.193†	0.564† / 0.589† / 0.249†	0.782 / 0.780	0.187 / 0.187
	10	0.730 / 0.801 / 0.179	0.705† / 0.771† / 0.189†	0.719† / 0.787† / 0.185†	0.590† / 0.607† / 0.240†	0.792 / 0.787†	0.181 / 0.183†
TB 2.0	9	0.868 / 0.945 / 0.093	0.860† / 0.937† / 0.100†	0.777† / 0.861† / 0.154†	0.681† / 0.737† / 0.208†	0.945 / 0.941†	0.092 / 0.096†
	18	0.897 / 0.960 / 0.076	0.890† / 0.956† / 0.080†	0.813† / 0.900† / 0.134†	0.705† / 0.767† / 0.204†	0.961 / 0.960†	0.075 / 0.075†

Table C.2: **Predicting whether a test agent solves each task it has not run**, after it runs a tenth or a fifth of the bank; mean over 30 splits. Columns 3–6 score each method on the tasks it did not administer, the quantity an operator faces. The last two columns score ours and FLUID on the tasks that neither administered, which isolates the predictor from the selection. †/*: significantly worse/better than ours.

A small adaptive sample predicts the rest of the bank well. On SWE-bench Verified, 50 tasks predict the outcome of the other 450 with accuracy 0.888 and AUC 0.953; 100 tasks give 0.909 and 0.964. On SWE-bench Lite, 30 tasks give 0.885 and 0.943.

Our predictions are better than FLUID's on 22 of 24 comparisons, with one loss. Our accuracy, AUC and Brier score are significantly better in 22 of the 24 comparisons, significantly worse in one, and tied in one. The two exceptions are the Brier score on τ-bench airline with five tasks and the AUC on CORE-bench Hard with nine tasks. The gains are small on the large banks, 0.003 to 0.011 in accuracy, and larger on CORE-bench Hard, up to 0.030. Predicting from the task's pass rate alone, or from the agent's pass rate alone, is significantly worse everywhere.

Scored on the tasks neither method administered, the advantage in ordering shrinks and the advantage in calibration remains. The comparison above scores each method on the tasks it did not run, which is the quantity an operator faces but not a comparison of predictors: adaptive selection changes which tasks remain, and the area under the curve is sensitive to that change. We therefore also score both methods on the tasks that neither administered (Table C.2, last two columns). The single area-under-the-curve loss disappears, CORE-bench Hard at nine tasks becoming 0.888 against 0.889; two accuracy cells turn significantly against us by 0.001 on SWE-bench Lite and one on Verified; and the Brier score stays significantly better in every cell except τ-bench airline at five tasks, where it is tied. We report both scorings rather than choosing the one that favours us, and read the marginal ordering differences as inconclusive.

Our probabilities are also better calibrated. With the shrinkage strength chosen per pool (Section 3.2), our expected calibration error is lower than FLUID's on SWE-bench Verified (0.010 against 0.012 at 50 tasks), SWE-bench Lite (0.014 against 0.020) and CORE-bench Hard (0.041 against 0.062), and tied on τ-bench airline (0.058 against 0.056). A fixed shrinkage strength lost two of these cells; Section D.1 reports both settings.

C.1.3 TASKS NEEDED UNDER ONE GUARANTEE FOR EVERY METHOD

The default rule of Section 3.4 is calibrated empirically. This section replaces its binned factor by a single multiplier c, stopping at the first K with $c\,u_i(K) \le \varepsilon$, and sets c by conformal risk control (Angelopoulos et al., 2024). For a calibration agent i the whole administration is replayed under c, and the loss is

$$L_i(c) = \mathbf{1}\big\{\,|\hat{s}_i(K_i(c)) - s_i| > \varepsilon\,\big\},$$

where $K_i(c)$ is that agent's own stopping time. The loss is made non-increasing in c by a running maximum from the right, $c = \infty$ means running the whole bank, and with n calibration agents

$$\hat{c} = \min\Big\{c : \tfrac{1}{n+1}\big(\textstyle\sum_{i=1}^{n} L_i(c) + 1\big) \le \alpha\Big\}.$$

Because the stopping time is part of the loss, its dependence on the data does not void the statement, as in risk-controlled early exiting and in risk-constrained stopping for sequential agents (Ringel et al., 2024; Jazbec et al., 2024; Wu & Rus, 2026): if the calibration agents and the test agent are exchangeable and the item fit has seen none of them, the test agent's error at its stopping time exceeds

Table C.3: **Tasks needed for the same guarantee.** Every method is calibrated by the same conformal risk control on the same held-out agents; parentheses give the realized share within ε. $> K$: the guarantee is not met within the method's evaluated grid, which ends at K. †: significantly more tasks than TITRATE.

Bank	$1-\alpha$	ε	TITRATE	Ours, fixed K	FLUID	ATLAS	CTT panel	PTA-IRT	tinyBench.
Verified (500)	0.95	4	**179** (0.96)	176	192	192	401†	>250	>100
		3	**227** (0.96)	234	274†	257†	441†	>250	>100
		2	**283** (0.96)	306	371†	345†	≥472†	>250	>100
Lite (300)	0.95	4	**121** (0.97)	130	143	165†	232†	>150	>60
		3	**154** (0.97)	158	197†	207†	260†	>150	>60
		2	**202** (0.96)	196	235†	≥239†	284†	>150	>60
CORE (45)	0.90	4	**32** (0.95)	34	36†	≥39†	41†	>22	>9
		3	**34** (0.95)	35	38†	≥40†	43†	>22	>9
		2	**39** (0.97)	40	42†	≥43†	≥43†	>22	>9
τ-bench (50)	0.90	4	**39** (0.93)	40	45†	45†	47†	>25	>10
		3	**42** (0.93)	43	49†	47†	48†	>25	>10
		2	**43** (0.94)	44	50†	48†	>48	>25	>10

ε with probability at most α. The statement is marginal over agents, not conditional on one agent, and $\alpha = 0.05$ requires $n \geq 19$.

No baseline offers such a statement, and the main comparison does not impose one (Table C.13). As a sensitivity check, the same calibration can give every baseline the same statement without changing anything inside it. A baseline estimates the score from a fixed number of tasks, so the calibrated parameter is that number, on the grid of budgets the baseline was evaluated at, with the loss monotonized in the same way. We extended ATLAS's official score estimate to a grid that runs to the whole bank in steps of 5%; FLUID and the CTT panel already run to the whole bank, and PTA-IRT and tinyBenchmarks were evaluated up to half and a fifth of it. Item parameters, λ and the default rule's replay use the training agents only; the conformal multiplier is calibrated on the other held-out agents of the same split, leave-one-out over the test agents, which no method's fit has seen, so the guarantee is exact for every method (in deployment, a set of recent agents withheld from the item fit); a split holds 44 test agents on Verified and 21 on Lite, which supports $\alpha = 0.05$, and 17 and 12 on the two small banks, which support $\alpha = 0.10$. The rule of thumb behind these sizes is the one the code applies: n calibration agents support a level α only if $n \geq \lceil 1/\alpha \rceil - 1$, because with fewer agents even the whole bank leaves the bound $(0+1)/(n+1)$ above α. A level that a bank cannot support is not reported, so the two small banks appear only at $\alpha = 0.10$. Within a supported level, an agent for which no multiplier meets the bound runs the whole bank and is charged for it. Terminal-Bench 2.0 is omitted because its reference score averages five attempts, so no estimator that sees one label per task converges to it and the statement is unattainable for every method.

TITRATE needs fewer tasks than every baseline that reaches the guarantee, in all twelve cells (Table C.3); its own count fixed in advance under the same calibration is within a tenth of the rule either way, and below it in two cells. For a guaranteed 0.95 within four points it needs 179 tasks on Verified and 121 on Lite, against 192 and 143 for FLUID and 192 and 165 for ATLAS; at two points it needs 283 and 202 against 371 and 235 for FLUID. The difference is significant against ATLAS in eleven cells and against FLUID in ten. The realized share is 0.96 to 0.97 where 0.95 is guaranteed and 0.93 to 0.97 where 0.90 is. Table C.3 reports the comparison cell by cell rather than as a summary: a dagger marks every cell where a method needs significantly more tasks than TITRATE, and TITRATE's realized share is given in every cell. The guarantee costs tasks: the empirical rule stops at 112 and 63 tasks on the two large banks for a realized 0.94.

C.1.4 TASKS NEEDED TO REACH A TARGET RANKING QUALITY

A fixed share of the bank can be too small for any method to rank well. On CORE-bench Hard and τ-bench airline, 10% is four or five tasks. We therefore also ask how many tasks each method needs before its ranking of the test agents reaches a target mean Kendall τ of 0.80, 0.85, 0.90 or 0.95. For

Benchmark	Target	Random	FLUID	ATLAS	TITRATE
SWE-bench Verified	$\tau \geq 0.8$	$31 \pm 4^{\dagger}$	8 ± 1	9 ± 1	$\mathbf{8 \pm 1}$
	$\tau \geq 0.85$	$51 \pm 6^{\dagger}$	15 ± 2	$20 \pm 5^{\dagger}$	$\mathbf{15 \pm 2}$
	$\tau \geq 0.9$	$101 \pm 8^{\dagger}$	60 ± 17	$84 \pm 16^{\dagger}$	$\mathbf{45 \pm 13}$
	$\tau \geq 0.95$	$246 \pm 19^{\dagger}$	$300 \pm 44^{\dagger}$	n.r.†	$\mathbf{188 \pm 22}$
SWE-bench Lite	$\tau \geq 0.8$	$47 \pm 11^{\dagger}$	15 ± 4	$18 \pm 4^{\dagger}$	$\mathbf{13 \pm 4}$
	$\tau \geq 0.85$	$83 \pm 15^{\dagger}$	29 ± 9	$39 \pm 11^{\dagger}$	$\mathbf{25 \pm 6}$
	$\tau \geq 0.9$	$148 \pm 18^{\dagger}$	$84 \pm 24^{\dagger}$	$107 \pm 47^{\dagger}$	$\mathbf{48 \pm 6}$
	$\tau \geq 0.95$	$223 \pm 18^{\dagger}$	250 [188, n.r.]†	n.r.†	$\mathbf{107 \pm 26}$
CORE-bench Hard	$\tau \geq 0.8$	$21 \pm 3^{\dagger}$	13 ± 2	$15 \pm 2^{\dagger}$	$\mathbf{11 \pm 1}$
	$\tau \geq 0.85$	$27 \pm 2^{\dagger}$	19 ± 2	$23 \pm 6^{\dagger}$	$\mathbf{18 \pm 2}$
	$\tau \geq 0.9$	$34 \pm 2^{\dagger}$	$28 \pm 2^{\dagger}$	n.r.†	$\mathbf{23 \pm 1}$
	$\tau \geq 0.95$	$42 \pm 1^{\dagger}$	n.r.†	n.r.†	$\mathbf{32 \pm 1}$
τ-bench airline	$\tau \geq 0.8$	$32 \pm 3^{\dagger}$	n.r.†	n.r.†	$\mathbf{22 \pm 2}$
	$\tau \geq 0.85$	$37 \pm 2^{\dagger}$	n.r.†	n.r.†	$\mathbf{26 \pm 3}$
	$\tau \geq 0.9$	$42 \pm 3^{\dagger}$	n.r.†	n.r.†	$\mathbf{34 \pm 3}$
	$\tau \geq 0.95$	$48 \pm 1^{\dagger}$	n.r.†	n.r.†	$\mathbf{43 \pm 1}$
Terminal-Bench 2.0	$\tau \geq 0.8$	$16 \pm 2^{\dagger}$	5 ± 0	$6 \pm 0^{\dagger}$	$\mathbf{5 \pm 0}$
	$\tau \geq 0.85$	$26 \pm 3^{\dagger}$	8 ± 1	9 ± 1	$\mathbf{8 \pm 1}$
	$\tau \geq 0.9$	$44 \pm 4^{\dagger}$	$20 \pm 3^{\dagger}$	18 ± 4	$\mathbf{16 \pm 2}$
	$\tau \geq 0.95$	n.r.	n.r.	n.r.	**67 [48, n.r.]**

Table C.4: **Tasks needed to reach a target ranking quality, all four targets.** Columns and marks as in the two-target version.

each method we average τ over the splits at every budget up to the whole bank. We then read off the first budget at which this mean reaches the target, interpolating linearly between budgets. The same rule applies to every method, and no method's settings change. Each baseline runs through its replication package and ranks the test agents by the estimate that package outputs; our method ranks by the predicted full-bank score of Equation 3, and the random subset by the observed pass rate on the tasks it drew. We resample the splits to obtain 95% intervals, and we resample them jointly for all methods to test the paired difference. Table 4 in the main text reports the result.

Our method needs the fewest tasks in all 20 cells. Against the three baselines, it needs significantly fewer tasks in 45 of the 60 comparisons, and it never needs significantly more. To reach $\tau = 0.80$ on SWE-bench Lite it needs 13 tasks, against 15 for FLUID, 18 for ATLAS and 47 for the random subset. To reach $\tau = 0.90$ on Lite it needs 48 tasks, against 84, 107 and 148.

The gap to FLUID and ATLAS grows with the target. At $\tau = 0.80$ on SWE-bench Verified, the three adaptive methods need 8 to 9 tasks, and the difference from FLUID is not significant. At $\tau = 0.90$ FLUID needs 60 tasks and ATLAS 84, against our 45. At $\tau = 0.95$ our method needs 188 tasks on Verified and 107 on Lite; FLUID needs 300 on Verified and does not reliably reach the target on Lite, ATLAS does not reach it within the bank, and the random subset needs 246 and 223. A target of 0.95 is therefore attainable on the two large banks at roughly two fifths of the bank, and it lies above the agreement between two complete single-attempt evaluations of Terminal-Bench, which is 0.910 at five attempts per cell.

The small banks are close to exhausted at the highest targets. On τ-bench airline neither FLUID nor ATLAS reaches $\tau = 0.80$ within the bank of 50 tasks. Our method reaches it with 22 tasks and the random subset with 32; at $\tau = 0.95$ the two need 43 and 48 tasks, most of the bank. On CORE-bench Hard the corresponding counts are 32 and 42 of 45 tasks.

C.1.5 THE SAME RANKING QUALITY WITH FEWER TASKS

Matching a baseline's ranking quality costs us a fraction of its tasks. We ask how many tasks our method needs to reach the ranking quality that each baseline reaches with 10% or 20% of the bank. The comparison must not use the test agents' results to set our budget. We therefore let the certificate

Target: baseline at	Verified		Lite		CORE		τ-bench	
	10% (50)	20% (100)	10% (30)	20% (60)	10% (4)	20% (9)	10% (5)	20% (10)
Random	26%	40%	33%	30%	100%$^{+}$	49%	83%	59%
Random + IRT	30%	41%	30%	26%	100%$^{+}$	54%	83%	48%
FLUID	70%$^{-}$	64%$^{-}$	89%	69%	100%$^{+}$	84%	91%	55%
ATLAS	51%	51%	75%	60%	102%$^{+}$	76%	89%	58%

Table C.5: **Share of a baseline's budget our method needs to match its mean Kendall** τ at a tenth or a fifth of the bank, the baseline's task count in parentheses. Our budget is fixed by the certificate from training agents only. $^{+}/^{-}$: our τ is significantly higher/lower; unmarked: not significant.

choose the budget for each split from training agents only, with the baseline's mean τ as the target and the median of the internal errors as the margin. We then compare our τ at that budget with the baseline's τ at its budget on the same split. Table C.5 reports the share of the baseline's budget that our method needs.

Against random subsets, our method needs a quarter to a half of the tasks. To match the random subset at 10% on SWE-bench Verified, it uses 26% of the budget, and 40% to match it at 20%. Against ATLAS it uses 51% at both budgets. On SWE-bench Lite the shares against the two random subsets are 26% to 33%. The paired difference in τ is not significant in any of these cells.

Against FLUID and ATLAS the saving is smaller on most benchmarks. To match them at 20%, our method uses 55% to 84% of their budget on Lite, CORE-bench Hard and τ-bench airline, with no significant difference in τ. On Verified it uses 64% to 70% of FLUID's budget and falls short by 0.006 to 0.007, a small but significant gap. The two methods share our selection principle, so large savings cannot be expected. At 10% on CORE-bench Hard and τ-bench airline the baselines use 4 or 5 tasks and our minimum budget is 4, which leaves no room.

C.1.6 HOW MANY TRAINING AGENTS ARE NEEDED

A new benchmark has few training agents. We therefore shrink the training pool to its first m agents, keep the test agents fixed, and measure the error of the predicted full-bank pass rate at 10% of the bank (Table C.7). We call m sufficient when the paired error stays within 0.5 percentage points of the full pool, a margin fixed before the analysis.

Fifteen to thirty agents are enough for the score, and not for everything else. On SWE-bench Verified, 15 of the 130 training agents are sufficient; the error is 2.61 points with 15 agents and 2.55 with all 130. On SWE-bench Lite 20 of 63 suffice, and on Terminal-Bench 2.0 30 of 95. With 5 agents the error roughly doubles on the three large banks. On the small banks the criterion is harder to meet, because each split has only 12 or 17 test agents and the intervals are wide: τ-bench airline needs 20 of 35 and CORE-bench Hard all 49, although its mean error with 20 agents is already within 0.4 points of the full pool.

How many agents a benchmark needs depends on what it wants, and the answers differ by a factor of five. Three measurements in this paper answer that question and they do not agree, because they ask different things. For predicting the score at a fixed budget, 15 agents are within 2% of what 130 give. For beating a complete single-attempt pass on a repeated-attempt bank, the advantage becomes significant between 13 and 17 agents (Table D.11). For the number of tasks a target ranking quality costs, the curve is still falling at 80 agents on both SWE-bench banks: 16.0 tasks at 80 against 14.3 with all 130 on Verified. And for the discriminations to correlate at 0.4 between two halves of the pool, about 60 agents are needed on both Verified and Terminal-Bench 2.0. A benchmark with twenty training agents can therefore use our procedure to report scores, and should not expect its item parameters to be stable.

Few agents cost tasks, and the cost stops falling after about twenty. Table C.6 fixes the benchmark, the splits and the test agents, and changes only the training pool. With 5 agents our method needs 27 tasks on SWE-bench Verified to reach $\tau = 0.85$ and 46 on Lite; with 20 agents it needs 16 and 30; with the full pool 14 and 20. The error target behaves the same way: with 5 agents the mean curve does not reach 3 percentage points within 150 tasks on either bank, and with 20 agents it needs 33

		Tasks needed with m training agents					
Benchmark	Target	5	10	20	40	80	all
SWE-bench Verified	$\tau = 0.85$	27	19	16	17	16	14
	error ≤ 3 points	n.r.	46	33	30	29	30
SWE-bench Lite	$\tau = 0.85$	46	32	30	25	20	20
	error ≤ 3 points	n.r.	30	22	20	17	17

Table C.6: **How many training agents a leaderboard needs before the method pays off.** The benchmark, the splits and the test agents are fixed; only the training pool changes, taking the first m agents of the same order. n.r.: the mean curve does not reach the target within the 150 tasks this experiment administers.

	All	Tasks	Score error ($\times 100$) with m training agents						
Benchmark	agents	run	m=5	10	15	20	30	all	m^*
Verified	130	50	4.80	2.91	2.61	2.50	2.43	2.55	15
Lite	63	30	4.56	3.03	2.79	2.54	2.43	2.37	20
CORE	49	4	10.16	8.26	7.90	7.39	7.74	7.02	49
τ-bench	35	5	9.12	8.41	8.08	7.48	7.65	7.89	20
TB 2.0	95	9	8.05	5.68	4.95	4.55	4.32	4.27	30

Table C.7: **How many training agents the method needs**, at a tenth of the bank. Test agents are those of the main protocol; training agents are the first m of the same random order. m^*: the smallest m such that it and every larger m stay within 0.5 percentage points of the full pool.

and 22 tasks. A new benchmark therefore needs on the order of twenty training agents, not hundreds, and the tasks a bank needs are not small because the pool is large.

How many agents the stopping margin itself needs. The margin of Section 3.4 is learned by replaying the administration on internal splits of the training pool, and each replay needs enough agents on both sides: we require at least eight in the part that fits the model, which with a quarter held out means a pool of about twelve. Below that floor every internal split is skipped, no factor $c_{1-\alpha}$ is estimated, and the rule falls back on the model's own uncalibrated error. Lowering the floor to five agents does produce a factor from six agents, but a replay that small yields a factor so large that every agent runs to the task cap. All five banks we report clear the floor with margin, the thinnest at 35 training agents.

C.1.7 Predicting later agents from earlier ones

Random splits mix old and test agents. A benchmark instead receives agents in time order, and test agents tend to be stronger. We therefore sort the agents of SWE-bench Verified and Lite by submission date and use six rolling cutoffs: the earliest 50%, 55%, ..., 75% are training agents, and the next 25% are test agents. Every method sees the same six splits.

Score prediction stays best under the time shift. With 10% of the bank, our error is 2.31 percentage points on Verified and 3.67 on Lite, against 3.90 and 6.42 for the random subset. Over the 24 pairs of benchmark, budget (10% or 20%) and cutoff, our error is lower in all 24. Terminal-Bench 2.0 gives the same picture, with our error at 4.60 points against 10.53 for the random subset. ATLAS's estimator is again the closest baseline: over the five budgets of the three benchmarks our error is significantly lower in 9 of the 15 cells, tied in 6, and never significantly higher.

The task budget needed for a target score error also stays lowest. The lower half of Table C.8 repeats the analysis of Section C.1.1 on the six cutoffs. To keep the error of the predicted full-bank pass rate below 3 percentage points, our method needs 20 tasks on SWE-bench Verified, 42 on Lite and 15 on Terminal-Bench 2.0, against 47, 72 and 22 for ATLAS's estimator and 109, 108 and more than the whole bank for the random subset. Our method needs the fewest tasks in all nine cells, significantly fewer in 15 of the 18 comparisons, and never significantly more.

Benchmark	Target	Random	FLUID	ATLAS	TITRATE
Verified (time split)	$\tau \geq 0.8$	$52 \pm 23^{\dagger}$	$21 \pm 16^{\dagger}$	$33 \pm 34^{\dagger}$	$\mathbf{18 \pm 13}$
	$\tau \geq 0.85$	$81 \pm 37^{\dagger}$	$56 \pm 30^{\dagger}$	$88 \pm 85^{\dagger}$	$\mathbf{41 \pm 27}$
	$\tau \geq 0.9$	$162 \pm 86^{\dagger}$	$136 \pm 55^{\dagger}$	n.r.†	$\mathbf{96 \pm 40}$
	$\tau \geq 0.95$	$357 \pm 51^{\dagger}$	n.r.†	n.r.†	$\mathbf{224 \pm 47}$
Lite (time split)	$\tau \geq 0.8$	$71 \pm 8^{\dagger}$	$72 \pm 16^{\dagger}$	n.r.†	$\mathbf{49 \pm 8}$
	$\tau \geq 0.85$	95 ± 18	114 ± 26	n.r.†	$\mathbf{66 \pm 34}$
	$\tau \geq 0.9$	151 ± 33	182 [148, n.r.]†	n.r.†	$\mathbf{136 \pm 34}$
	$\tau \geq 0.95$	249 ± 26	n.r.†	n.r.†	$\mathbf{215 \pm 37}$
TB 2.0 (time split)	$\tau \geq 0.8$	$31 \pm 4^{\dagger}$	$16 \pm 3^{\dagger}$	23 [12, n.r.]†	$\mathbf{13 \pm 2}$
	$\tau \geq 0.85$	$44 \pm 11^{\dagger}$	$36 \pm 17^{\dagger}$	n.r.†	$\mathbf{19 \pm 4}$
	$\tau \geq 0.9$	76 [54, n.r.]†	n.r.†	n.r.†	$\mathbf{30 \pm 21}$
	$\tau \geq 0.95$	n.r.	n.r.	n.r.	n.r.

Benchmark	Target	Random	ATLAS	TITRATE
Verified (time split)	error ≤ 4	$49 \pm 22^{\dagger}$	$25 \pm 1^{\dagger}$	$\mathbf{11 \pm 1}$
	error ≤ 3	$109 \pm 20^{\dagger}$	$47 \pm 4^{\dagger}$	$\mathbf{20 \pm 4}$
	error ≤ 2	$146 \pm 17^{\dagger}$	$101 \pm 10^{\dagger}$	$\mathbf{87 \pm 8}$
Lite (time split)	error ≤ 4	$67 \pm 21^{\dagger}$	31 ± 10	$\mathbf{24 \pm 9}$
	error ≤ 3	$107 \pm 22^{\dagger}$	$72 \pm 10^{\dagger}$	$\mathbf{42 \pm 6}$
	error ≤ 2	n.r.†	$137 \pm 9^{\dagger}$	$\mathbf{68 \pm 7}$
TB 2.0 (time split)	error ≤ 4	n.r.†	$15 \pm 1^{\dagger}$	$\mathbf{10 \pm 1}$
	error ≤ 3	n.r.†	$22 \pm 2^{\dagger}$	$\mathbf{15 \pm 2}$
	error ≤ 2	n.r.	44 ± 2	$\mathbf{30 \pm 7}$

Table C.8: **Tasks per test agent needed under time splits**, six cutoffs. Top: to reach a target mean Kendall τ; bottom: to bring the error of the predicted full-bank pass rate below a target. Both read each method's mean curve and interpolate between budgets. $\pm$: half the width of the 95% interval from resampling cutoffs; bold: fewest; n.r.: not reached within the bank; †: ours needs significantly fewer tasks.

Ranking needs more tasks under the time shift, and our method still needs the fewest. Kendall τ among the later agents is lower for every method at the same budget. Repeating the tasks-to-target analysis on the six cutoffs (Table C.8, upper half), our method needs the fewest tasks in 11 of the 12 cells, and in the twelfth, $\tau = 0.95$ on Terminal-Bench 2.0, no method reaches the target within the bank: 18 tasks on Verified, 49 on Lite and 13 on Terminal-Bench 2.0 reach $\tau = 0.80$, against 21, 72 and 16 for FLUID; ATLAS does not reach 0.80 on Lite and needs 33 tasks on Verified. The paired difference is significant in 29 of the 33 comparisons and favours a baseline in none. At a fixed small budget the picture can differ: on Terminal-Bench 2.0 with nine tasks FLUID ranks ahead, 0.737 against our 0.690, though neither is near $\tau = 0.80$ there.

The certificate does not hold under the time shift. It predicts the achievable τ from the spread of the training agents' abilities, and that prediction does not transfer. We test two explanations in Appendix F and refute both by measurement; the later cohort is not more tightly bunched. The achievable τ is lower than predicted. At stated reliability 0.95, the target is met in only one to four of the six cutoffs, a realized reliability of 0.17 to 0.67 over the four targets and three benchmarks. Under random splits the same certificate reaches 0.77 to 1.00 (Table C.14). A benchmark that receives a new model generation therefore cannot use the certificate as it stands. Re-estimating the spread on the new cohort, or recalibrating on recent agents, is the natural repair; the recency experiment below shows that the second alone does not restore the stated level for the per-agent rule, and we have not validated either as a fix.

The per-agent stopping rule transfers better than the certificate, and not fully. Its margin is calibrated by replaying the procedure on the earlier agents, so it faces the same shift. Under the six cutoffs, at a four-point target, it stops after 56–69 tasks on SWE-bench Verified and puts 0.82–0.91 of the later agents within target (mean 0.85), after 54–70 tasks on Lite with 0.76–1.00 (mean 0.88), and

after 52–60 tasks on Terminal-Bench 2.0 with 0.75–0.97 (mean 0.88), against a stated 0.95 and the 0.91–0.97 realized under random splits. At three points the means are 0.86, 0.79 and 0.78. A count fixed at the rule's mean meets the target about as often (0.83, 0.86 and 0.96). The shortfall is concentrated at the earliest cutoffs, where the fewest and oldest agents calibrate.

A recent training pool predicts scores as well as the whole history, and a small random one does not. Under the same cutoffs we calibrate on the M most recent available agents, and, as a control for pool size, on M available agents drawn at random (Table C.9). For the score, the 20 to 30 most recent agents match every available agent: 2.38 against 2.31 points on Verified, 3.31 against 3.67 on Lite and 4.55 against 4.60 on Terminal-Bench 2.0 at a tenth of the bank. At the same pool size, recency matters where the shift is visible: the recent pool beats the random one by 0.5 points on Lite (3.31 against 3.79 at $M = 20$) and by 1.5 and 0.5 points on Terminal-Bench 2.0 ($M = 30$ and 60), while on Verified the three pools are within 0.2 points of each other. Table C.11 gives the error at every cutoff; it does not drift upward with time, and on Lite it falls as the pool grows. The stopping rule is insensitive to the pool, 0.84–0.92 within four points for every pool but the random 20 on Lite (0.80). The certificate is where the pools differ most, and mostly for the wrong reason: a recent pool of 30 reaches its target at every cutoff, but by fixing 160 tasks on Verified and 166 on Lite where the full history fixes 19 and 67, because a small pool widens the internal margin; the random pool of the same size moves in the same direction. Only Terminal-Bench 2.0 with $M = 60$ (27 tasks, every cutoff met, against 23 tasks and two thirds of the cutoffs) and Lite with the recent 40 (79 tasks, 0.67, against 67 tasks and 0.33) improve at a comparable cost. With six cutoffs, realized reliability moves in steps of one sixth, so these differences are indicative, not established.

The baselines refitted on the same windows tell the same story and fall further behind on the small pools (Table C.10). Recency helps every method on Lite, where the recent 20 beat the random 20 by 0.5 points for us, 1.0 for FLUID, 0.3 for ATLAS and 1.2 for the classical panel, and it helps us most on Terminal-Bench 2.0 (4.55 against 6.05 at $M = 30$). What the windows expose is the cost of a small pool for methods without shrinkage: ATLAS's p-IRT error rises from 2.88 with every earlier agent to 4.06 with the recent 30 on Verified and from 4.03 to 6.29 with the recent 20 on Lite, and the classical panel from 2.77 to 3.81 and from 3.74 to 4.58, while our error moves by at most 0.4 points and FLUID's by at most 0.5. Our method has the lowest error in 12 of the 15 pool–budget cells at a tenth of the bank and in 13 at a fifth; where FLUID is ahead the margin is at most 0.4 points. PTA-IRT, rerun on each pool in its no-trajectory mode, sits between FLUID and ATLAS on the SWE-bench pools (2.68–4.16 points on Verified, 3.70–5.16 on Lite) and fails on Terminal-Bench 2.0, where its four-parameter fit on majority labels gives 9.5 points with every earlier agent and 11–17 on the windows. The random subset needs no calibration and is the same in every row, 3.90, 6.42 and 10.53 points at a tenth.

C.1.8 A BUDGET FIXED BEFORE THE AGENT RUNS

ATLAS is the only baseline whose released code decides its own budget: it stops each agent when the standard error of its ability falls below ϵ, after at least 30 items. At the thresholds of its paper, $\epsilon = 0.3$, 0.2 and 0.1, it reaches $\tau = 0.838$, 0.839 and 0.863 on SWE-bench Lite with 37.0, 39.9 and 102.3 tasks, and 0.866, 0.863 and 0.859 on CORE-bench Hard with 33.6 to 40.1 tasks. The threshold controls precision, not ranking: a tighter threshold nearly triples the cost on Lite for 0.025 in τ, and on CORE-bench Hard it buys nothing. The 30-item floor also exceeds 10% of every bank except SWE-bench Verified and Lite.

Scored on the full-bank pass rate with its own p-IRT estimator, the same rule shows why a threshold on ability precision cannot serve as a target (Table C.12). At $\epsilon = 0.3$ and 0.2 the rule stops at the 30-item floor for nearly every agent on SWE-bench Verified, Lite and Terminal-Bench 2.0, so the threshold is not what decides the budget there, and the share of agents within four points is 0.70–0.73 on Verified and Lite. Tightening to $\epsilon = 0.1$ spends 104 tasks on Lite for 0.82 within four points, where our rule spends 63 for 0.94, and 48 tasks on Verified for 0.73, where our rule spends 112 for 0.94. On the two small banks the tightest threshold reaches 0.96 and 0.93 within four points, but by running 40 of 45 and 46 of 50 tasks; our rule reaches 0.95 and 0.97 with 30 and 41. The same threshold therefore delivers anywhere from 0.59 (τ-bench airline at $\epsilon = 0.3$) to 0.89 (CORE-bench Hard at $\epsilon = 0.3$) of agents within four points, and nothing in the rule tells the operator which. Terminal-Bench 2.0 with majority labels is the closest case: $\epsilon = 0.1$ spends 74 tasks for 0.90 against our 64 for 0.91. The practical reading is that ability-precision stopping suffices only where it runs most of the bank.

Bank	Calibration pool	Score MAE 10%	Score MAE 20%	Stopping, $\varepsilon = 4$ tasks	Stopping, $\varepsilon = 4$ met	Certificate $\tau^* = 0.85$ tasks	Certificate $\tau^* = 0.85$ met	Certificate $\tau^* = 0.90$ tasks	Certificate $\tau^* = 0.90$ met
SWE-bench Verified	all earlier	2.31	1.83	61	0.85	19	0.17	53	0.33
SWE-bench Verified	recent 30	2.38	1.69	66	0.84	160	1.00	328	1.00
SWE-bench Verified	random 30	2.27	1.70	82	0.86	50	0.83	142	0.83
SWE-bench Verified	recent 60	2.42	1.70	76	0.86	51	0.50	175	0.83
SWE-bench Verified	random 60	2.25	1.71	61	0.87	30	0.67	81	0.67
SWE-bench Lite	all earlier	3.67	2.16	60	0.88	67	0.33	94	0.33
SWE-bench Lite	recent 20	3.31	2.09	69	0.87	166	1.00	259	1.00
SWE-bench Lite	random 20	3.79	2.58	96	0.80	154	0.83	193	0.83
SWE-bench Lite	recent 40	3.53	2.05	68	0.90	79	0.67	140	0.50
SWE-bench Lite	random 40	3.71	2.23	59	0.80	72	0.67	112	0.17
Terminal-Bench 2.0	all earlier	4.60	2.61	56	0.88	23	0.67	68	0.67
Terminal-Bench 2.0	recent 30	4.55	2.72	58	0.91	79	1.00	89	0.83
Terminal-Bench 2.0	random 30	6.05	3.12	53	0.91	66	1.00	89	0.67
Terminal-Bench 2.0	recent 60	4.56	2.45	53	0.90	27	1.00	88	0.67
Terminal-Bench 2.0	random 60	5.09	2.85	51	0.92	30	1.00	82	0.67

Table C.9: **Does a recent training pool transfer better across time?** At each of six cutoffs the earliest 50–75% of agents by submission date are available and the next 25% are new. All earlier: every available agent calibrates. Recent M: the M most recent. Random M: M drawn at random. Score MAE in percentage points; stopping and certificate columns state a reliability of 0.95. Means over the six cutoffs.

		10% of the bank							20% of the bank						
Bank	Calibration pool	Ours	FLUID	ATLAS (p-IRT)	PTA-IRT	CTT panel	Mid-range	Random	Ours	FLUID	ATLAS (p-IRT)	PTA-IRT	CTT panel	Mid-range	Random
SWE-bench Verified	all earlier	**2.31**	2.40	2.88	2.98	2.77	3.21	3.90	**1.83**	1.95	2.01	2.23	2.08	2.64	3.26
SWE-bench Verified	recent 30	**2.38**	2.40	4.06	3.45	3.81	3.78	3.90	1.69	**1.67**	3.19	2.72	3.07	2.96	3.26
SWE-bench Verified	random 30	**2.27**	2.35	4.03	4.16	3.02	4.01	3.90	**1.70**	1.86	2.92	3.69	2.49	3.16	3.26
SWE-bench Verified	recent 60	2.42	**2.32**	3.40	2.68	3.48	3.41	3.90	**1.70**	1.75	2.59	2.27	2.85	2.62	3.26
SWE-bench Verified	random 60	**2.25**	2.48	3.28	2.96	2.89	3.56	3.90	**1.71**	2.03	2.27	2.29	2.10	2.72	3.26
SWE-bench Lite	all earlier	**3.67**	3.87	4.03	3.70	3.74	4.51	6.42	**2.16**	2.85	3.22	2.80	2.99	3.77	4.18
SWE-bench Lite	recent 20	**3.31**	3.37	6.29	4.04	4.58	3.79	6.42	**2.09**	2.39	5.38	3.72	3.92	3.02	4.18
SWE-bench Lite	random 20	**3.79**	4.36	6.61	5.16	5.73	5.56	6.42	**2.58**	3.11	5.64	4.84	5.13	4.57	4.18
SWE-bench Lite	recent 40	3.53	**3.14**	5.82	3.90	4.25	4.27	6.42	**2.05**	2.30	4.74	3.51	3.64	3.17	4.18
SWE-bench Lite	random 40	3.71	**3.39**	4.94	4.39	3.96	4.69	6.42	**2.23**	2.47	3.83	3.54	3.20	3.79	4.18
Terminal-Bench 2.0	all earlier	**4.60**	5.63	5.22	9.53	7.38	10.89	10.53	**2.61**	3.45	3.29	9.24	5.29	10.55	7.98
Terminal-Bench 2.0	recent 30	**4.55**	5.62	6.79	14.80	7.22	7.88	10.53	**2.72**	3.52	5.15	13.88	5.44	6.11	7.98
Terminal-Bench 2.0	random 30	6.05	**5.72**	6.91	17.48	7.07	10.99	10.53	**3.12**	3.18	4.91	16.92	5.78	9.86	7.98
Terminal-Bench 2.0	recent 60	**4.56**	6.21	4.82	11.47	7.21	11.08	10.53	**2.45**	3.64	3.41	10.81	4.86	8.99	7.98
Terminal-Bench 2.0	random 60	**5.09**	6.47	5.13	13.42	7.98	11.89	10.53	**2.85**	4.02	3.51	13.15	5.68	10.33	7.98

Table C.10: **Every method under the same training windows.** Score MAE in percentage points for the later agents, mean over the six chronological cutoffs: all earlier agents, the M most recent, or M at random. Every baseline is refitted on each pool by its released code; the random subset needs no calibration. Bold: best in the row.

The one bank where a threshold of ATLAS's paper leaves at least 0.95 of agents within four points is CORE-bench Hard, at $\epsilon = 0.1$, and there it runs 40 of 45 tasks. Its tightest threshold leaves 0.93 on τ-bench airline and 0.90 on Terminal-Bench 2.0. On SWE-bench Verified and Lite no threshold leaves more than 0.73 and 0.82. A score target is therefore essential on banks of a few hundred tasks. On a bank of a few dozen tasks, a tight ability threshold is close to running the whole bank and can serve.

The two criteria do not agree, and the disagreement is the point. On τ-bench airline ATLAS at its tightest threshold has the lower mean error, 0.80 points against our 1.48, and yet leaves more agents outside the four-point band, 0.93 against 0.97; on CORE-bench Hard its mean error is 0.48 against our 1.49 at 40 tasks against our 30. A mean error rewards a rule that is very accurate on typical agents and badly wrong on a few, and it falls whenever a rule simply runs more tasks. Our rule is calibrated on the tail of the error at a stated level, so it is uniformly moderate rather than usually sharp, and it stops as soon as that level is met instead of continuing to drive the mean down. We therefore report the share within the tolerance as the primary quantity, because that is the statement the rule makes, and give the mean error here so that both readings are available.

The two rules above stop at different counts. Table C.13 removes that difference: each test agent receives exactly the number of tasks at which ATLAS stopped for that agent, chosen by our own selection, and both estimates are scored against the full-bank pass rate. Nothing is imposed on ATLAS

Bank	Calibration pool	Cutoff (share of agents available for calibration) 50%	55%	60%	65%	70%	75%
SWE-bench Verified	all earlier	2.06	1.95	2.38	2.43	2.63	2.42
SWE-bench Verified	recent 30	1.99	3.19	2.63	2.42	1.93	2.08
SWE-bench Verified	random 30	2.27	2.14	2.49	2.29	2.53	1.90
SWE-bench Verified	recent 60	2.42	2.79	2.98	2.06	2.25	2.04
SWE-bench Verified	random 60	1.97	1.89	2.47	2.44	2.41	2.32
SWE-bench Lite	all earlier	3.89	3.88	4.18	3.88	3.40	2.80
SWE-bench Lite	recent 20	3.67	4.20	3.55	2.67	2.82	2.94
SWE-bench Lite	random 20	5.93	3.72	3.37	3.99	2.47	3.25
SWE-bench Lite	recent 40	3.49	3.80	4.09	3.20	3.22	3.38
SWE-bench Lite	random 40	2.88	4.34	5.46	3.70	2.91	2.97
Terminal-Bench 2.0	all earlier	4.61	4.71	4.59	4.40	4.96	4.32
Terminal-Bench 2.0	recent 30	3.70	4.08	4.21	4.47	5.00	5.88
Terminal-Bench 2.0	random 30	5.59	5.32	5.75	7.19	5.77	6.70
Terminal-Bench 2.0	recent 60	4.47	4.51	4.48	3.98	5.67	4.26
Terminal-Bench 2.0	random 60	5.74	4.46	4.74	4.29	5.82	5.46

Table C.11: **Score error over time.** Mean absolute error in percentage points at a tenth of the bank for the next 25% of agents after each cutoff, by training pool.

and no budget is read in hindsight. TITRATE has the lower score error in all twelve cells of the four single-attempt banks, and a significantly higher share within four points in eleven; on Terminal-Bench 2.0 ATLAS's error is lower by 0.14 to 0.18 points at the two looser thresholds.

The certificate instead takes the target itself. For $\tau^* = 0.85$ with stated reliability 0.99, it fixes on average 32.7 tasks on SWE-bench Verified and 64.8 on Lite before any test-agent run, and the target is met in 0.90 and 0.87 of the splits.

Two claims have to be separated here. The first is that a budget can be fixed in advance at all. That claim holds by construction: the number of tasks is computed from the training agents only, so an operator can price the evaluation before the first run, which neither baseline supports, since FLUID takes the budget as an input and ATLAS decides it while running. The second claim is that the stated reliability is the reliability delivered, and that one is only approximately true. Table C.14 gives the realized reliability at stated levels 0.95 and 0.98. At 0.95 it ranges from 0.77 to 0.97 over the 20 benchmark-by-target cells, averages 0.885, and falls short of the stated level in 15 of them, by up to 0.18. Thirty internal splits estimate a high quantile coarsely, which would produce this, but we did not test that it is the cause. At 0.98 it ranges from 0.80 to 1.00, averages 0.929, and reaches 0.90 or above in 18 of the 20 cells, for a median of 12% more tasks and at most 20% more.

The practical reading is that reliability is a dial with a price, and the price is measurable in advance from the same training agents. We therefore recommend stating 0.98 when the delivered level matters, and we report the realized level rather than the stated one throughout. What the certificate cannot do is survive a change of generation: under our time splits, where the test agents are later and stronger than the training pool, the realized reliability falls to between 0.17 and 0.67 (Section C.1.7). Calibrating the internal splits chronologically as well recovers it only partially, to 0.83 on Terminal-Bench 2.0 and 0.67 on SWE-bench Lite, at 1.2 to 2.9 times the tasks.

C.2 A BENCHMARK WITH REPEATED ATTEMPTS: TERMINAL-BENCH 2.0

On Terminal-Bench 2.0 the unit of cost is a run, and a task can be run up to five times. The baselines' released code accepts one binary outcome per task. They can therefore run each selected task once, or five times as the protocol requires and reduce the attempts to a majority label.

C.2.1 RUNS NEEDED AGAINST EVERY BASELINE

Counted in runs rather than tasks, our procedure is ahead of every baseline at every budget except one complete pass at 89 runs, where uniform reruns tie it (0.928 against 0.927). Figure C.2 compares every method by the number of runs per test agent. Three findings stand out.

Bank (tasks)	Rule	Threshold	Tasks [10–90%]	MAE	≤ 4 pt	≤ 3 pt	≤ 2 pt
SWE-bench Verified (500)	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.3$	31 [30, 30]	3.05	0.70	0.57	0.42
	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.2$	39 [30, 30]	2.97	0.71	0.59	0.44
	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.1$	48 [30, 38]	2.87	0.73	0.61	0.46
	Ours	$\varepsilon = 4$ points	112 [50, 161]	1.67	0.94	–	–
	Ours	$\varepsilon = 3$ points	170 [114, 200]	1.29	–	0.93	–
	Ours	$\varepsilon = 2$ points	191 [177, 200]	1.23	–	–	0.81
SWE-bench Lite (300)	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.3$	38 [30, 30]	2.95	0.72	0.59	0.42
	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.2$	41 [30, 30]	2.88	0.73	0.60	0.43
	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.1$	104 [30, 260]	2.04	0.82	0.74	0.62
	Ours	$\varepsilon = 4$ points	63 [53, 70]	1.75	0.94	–	–
	Ours	$\varepsilon = 3$ points	105 [88, 119]	1.16	–	0.94	–
	Ours	$\varepsilon = 2$ points	148 [125, 166]	0.74	–	–	0.94
CORE-bench Hard (45)	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.3$	34 [30, 42]	1.68	0.89	0.82	0.69
	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.2$	37 [30, 42]	1.21	0.90	0.86	0.77
	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.1$	40 [37, 42]	0.48	0.96	0.95	0.91
	Ours	$\varepsilon = 4$ points	30 [23, 35]	1.49	0.95	–	–
	Ours	$\varepsilon = 3$ points	33 [28, 38]	1.05	–	0.95	–
	Ours	$\varepsilon = 2$ points	38 [34, 41]	0.47	–	–	0.97
τ-bench airline (50)	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.3$	34 [30, 48]	3.95	0.59	0.50	0.39
	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.2$	42 [30, 49]	1.85	0.82	0.77	0.71
	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.1$	46 [37, 49]	0.80	0.93	0.90	0.85
	Ours	$\varepsilon = 4$ points	41 [39, 42]	1.48	0.97	–	–
	Ours	$\varepsilon = 3$ points	44 [42, 45]	0.99	–	0.97	–
	Ours	$\varepsilon = 2$ points	45 [44, 46]	0.49	–	–	0.98
Terminal-Bench 2.0 (89)	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.3$	31 [30, 30]	2.36	0.81	0.68	0.51
	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.2$	37 [30, 87]	2.39	0.80	0.67	0.51
	ATLAS	$\mathrm{SE}(\hat{\theta}) \leq 0.1$	74 [30, 89]	1.99	0.90	0.78	0.58
	Ours	$\varepsilon = 4$ points	64 [51, 75]	1.86	0.91	–	–
	Ours	$\varepsilon = 3$ points	83 [75, 88]	1.79	–	0.83	–
	Ours	$\varepsilon = 2$ points	89 (all) [89, 89]	1.85	–	–	0.62

Table C.12: **ATLAS's own stopping rule, scored on the full-bank pass rate.** It stops an agent once the standard error of its ability falls below the threshold, after at least 30 items, and is scored by its own p-IRT estimator. Ours is the calibrated rule at the target it is asked for, stated reliability 0.95, so its share is reported only at that target. Tasks: mean count [10th–90th percentile across agents]. MAE in percentage points.

With few runs, one run per task beats the protocol's five. Every method ranks test agents better when it spends a budget on more tasks run once than on fewer tasks run five times. At 45 runs, FLUID reaches $\tau = 0.903$ with 45 tasks run once and 0.858 with 9 tasks run five times; our method reaches 0.909 and 0.858. Below 89 runs our method, run once per task, is the best in every column.

Deciding reruns beats every baseline at every budget above the bank. After all 89 tasks are run once, our method spends further runs where the model expects them to help (Section C.2.3). With 134 runs it reaches $\tau = 0.951$, with 178 runs 0.965 and with 356 runs 0.992.

The fair comparison gives the baselines the mean pass rate. Reducing five attempts to a majority label discards information, so we also pass the attempt-averaged pass rate wherever the released code accepts it. Only the random subset's code does. FLUID's engine rejects any response other than 0 or 1 and ATLAS's adaptive test uses a dichotomous model, so neither can use repeated attempts. With the mean pass rate, the random subset of the protocol is the strongest baseline. It reaches 0.900 with 134 runs, 0.920 with 178, 0.952 with 267 and 0.970 with 356. Our method is ahead by 0.051, 0.045, 0.027 and 0.022 at these budgets. The random subset needs 267 runs to reach the 0.951 that our method reaches with 134, about twice as many. With all 445 runs it reproduces the official ranking, because it then is the official protocol.

Table C.13: **ATLAS's own stopping rule, and TITRATE given the same tasks.** Each test agent receives exactly the number of tasks at which ATLAS stopped for that agent. Bold: significantly better (paired bootstrap over splits, 95%).

			Within 4 points		MAE (points)	
Bank	$\mathrm{SE}(\hat{\theta}) \leq$	Tasks	ATLAS	TITRATE	ATLAS	TITRATE
Verified (500)	0.3	31	0.70	**0.75**	3.05	**2.76**
	0.2	39	0.71	**0.75**	2.97	**2.75**
	0.1	48	0.73	**0.77**	2.87	**2.67**
Lite (300)	0.3	38	0.72	**0.80**	2.95	**2.33**
	0.2	41	0.73	**0.80**	2.88	**2.31**
	0.1	104	0.82	**0.89**	2.04	**1.54**
CORE (45)	0.3	34	0.89	**0.94**	1.68	**1.20**
	0.2	37	0.90	**0.94**	1.21	**0.91**
	0.1	40	0.96	0.98	0.48	**0.35**
τ-bench (50)	0.3	34	0.59	**0.81**	3.95	**2.33**
	0.2	42	0.82	**0.91**	1.85	**1.13**
	0.1	46	0.93	**0.98**	0.80	**0.50**
TB 2.0 (89)	0.3	31	0.81	0.79	**2.36**	2.54
	0.2	37	0.80	0.79	**2.39**	2.53
	0.1	74	0.90	0.90	1.99	2.02

Benchmark (test agents)	$\tau^* = 0.75$	0.80	0.85	0.90
TB 2.0 (32)	0.97 / 0.97 (5 / 6)	0.97 / 1.00 (8 / 9)	0.83 / 0.90 (14 / 15)	0.87 / 0.97 (33 / 38)
Verified (44)	0.90 / 0.93 (8 / 8)	0.80 / 0.90 (12 / 14)	0.93 / 0.97 (26 / 31)	0.77 / 0.90 (100 / 112)
Lite (21)	0.90 / 0.93 (22 / 26)	0.97 / 0.93 (34 / 39)	0.87 / 0.90 (52 / 57)	0.90 / 1.00 (92 / 107)
CORE (17)	0.87 / 0.90 (15 / 16)	0.83 / 0.93 (18 / 20)	0.87 / 0.90 (24 / 27)	0.87 / 0.93 (32 / 34)
τ-bench (12)	0.97 / 0.97 (34 / 37)	0.97 / 0.97 (39 / 40)	0.87 / 0.87 (42 / 44)	0.77 / 0.80 (45 / 46)

Table C.14: **The budget certificate with its calibrated margin.** Each cell: realized reliability over 30 held-out splits at a stated 0.95 and 0.98, and in parentheses the mean number of tasks fixed in advance.

Allowed to spend the attempts individually, our score error on Terminal-Bench 2.0 is one half to one third of ATLAS's at the same number of runs. A baseline that is fed a majority label consumes the five recorded attempts of the cell but cannot use them separately. Charging every method the runs it consumes, ATLAS's estimator reaches 3.04, 2.47 and 1.96 percentage points of score error at 90, 135 and 220 runs, where our procedure reaches 2.05, 1.36 and 0.76 at 89, 134 and 223. The 1.96 points it needs 220 runs for we reach with about 120. This is the same budget spent differently, not a different budget: the gain comes from using the attempts rather than their majority.

Baselines fed majority labels cannot recover the official ranking even with every run. With all 445 runs, FLUID reaches $\tau = 0.945$ and ATLAS 0.929. Uniform reruns with the attempt-averaged score trail our allocation by 0.010 to 0.012 at every budget from 134 to 356 runs. All of these differences are significant: against the random subset with the mean pass rate, our method is ahead on 29 or 30 of the 30 splits at each budget, and every paired 95% interval excludes zero; against uniform reruns the paired differences are 0.010 to 0.013.

C.2.2 PREDICTING RERUNS

A test agent runs each of the 89 Terminal-Bench 2.0 tasks once. Each model estimates its ability from these runs and predicts the remaining archived attempts of every cell. The left half of Table C.15 reports the results over 30 splits.

The binomial model treats attempts as independent given ability. It predicts that an agent that passed a task passes the next attempt with probability 0.752. The observed rate is 0.823. It predicts that an agent that failed passes the next attempt with probability 0.212; the observed rate is 0.151. The

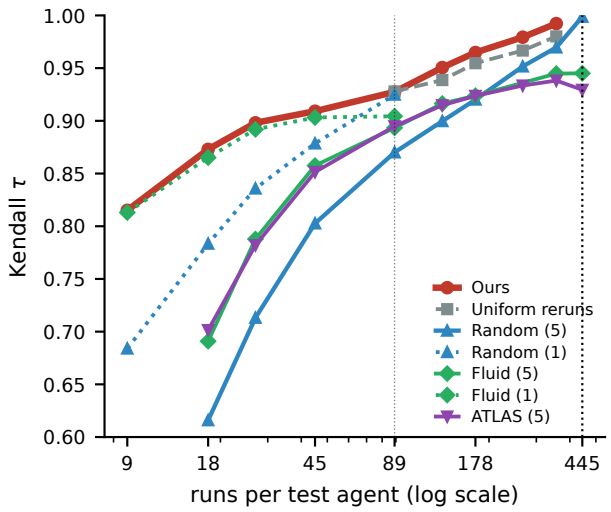

Figure C.2: **Ranking quality against runs per test agent, Terminal-Bench 2.0.** Our procedure runs each task once up to 89 runs and then decides which cells to rerun. Dotted lines run each task once, solid lines five times; the official protocol spends 445 runs.

	Next attempt of a cell run once			Cell not run: always pass / always fail / mixed				
	Pr(pass | passed)	Pr(pass | failed)	Log-lik. per attempt	Log loss	Accuracy (%)	Predicted share (%) pass	fail	mixed
Observed	0.823	0.151	–	–	–	29.2	37.6	33.2
Task's historical frequencies	–	–	−0.534	0.855	59.3	29.0	38.2	32.7
Binomial 2PL (independent attempts)	0.752	0.212	−0.363	0.713	71.2	22.5	31.0	46.4
Repeat-aware IRT (ours)	**0.818**	**0.156**	**−0.318**	**0.591**	**73.1**	28.7	36.7	34.6

Table C.15: **Predicting repeated attempts on Terminal-Bench 2.0**, mean over 30 splits. Left: a test agent runs each of the 89 tasks once and each model predicts the remaining archived attempts of every cell. Right: the agent runs every other task once and each model classifies the unrun cell. –: not defined for that row.

repeat-aware model predicts 0.818 and 0.156. Its log-likelihood per held-out attempt is higher on every one of the 30 splits, by 0.038 to 0.054. Averaged over splits, the median interaction scale σ_j is 1.49 logits. One standard deviation of ability, multiplied by the median discrimination of 2.29, spans 2.29 logits. Agent–task interaction is thus of the same order as the spread in ability, and a single run reveals much of it.

C.2.3 Deciding reruns

Table C.19 reports the pre-registered experiment. All three criteria hold.

Our allocation beats uniform reruns at every budget. Its score error is lower by 0.0033 to 0.0036 at 134, 178, 223, 267 and 356 runs, and every paired interval excludes zero. At 267 runs, which uniform allocation spends as three runs per task, our error is 0.0054 against 0.0087.

It saves a fifth of the runs (214 against 267). With 214 runs our error is 0.0081, below the 0.0087 that uniform allocation reaches with 267 runs; the paired difference is -0.0006 $[-0.0011, -0.0001]$. With 267 runs our error is already below that of uniform allocation with 356 runs. With 356 runs our error is 0.0020, and Kendall τ with the official ranking is 0.992.

The gain comes from the model of repeated attempts. Allocation by the binomial model, which treats attempts as independent, is significantly worse than ours at all five budgets. Sending runs to cells that have already shown both outcomes, the obvious heuristic, is worse than uniform allocation.

Allocation	134	214	267	356
Uniform, cell mean	1.72†	1.16†	0.87†	0.55†
Uniform, model estimate	1.62†	1.12†	0.87†	0.55†
Mixed cells first	1.78†	1.24†	0.99†	0.51†
Binomial 2PL	1.64†	0.95†	0.67†	0.24†
TITRATE	**1.36**	**0.81**	**0.54**	**0.20**

Table C.16: **Score error at fixed run budgets on Terminal-Bench 2.0.** Columns: runs per test agent, after one pass over the 89 tasks. †: significantly higher than TITRATE (Table C.19).

	N runs		$2N$ runs	
Release	Pass	TITRATE	Pass	TITRATE
TB 2.0	2.76	2.28	2.13	2.13
TB 2.1	2.64	2.38	2.11	2.11
TB 4.0	3.41	3.30	2.69	2.69

Table C.17: **Prediction of an independent evaluation.** Error against three held-out attempts per cell, acquiring from the other two; Pass: complete passes. At $2N$ both exhaust the pool (Table D.10).

Changing only the estimator while keeping uniform allocation helps at 134 runs and not after. The saving therefore comes from where the runs go.

Close agents are separated better. Among test agents whose official scores differ by less than 0.03, our procedure orders 0.855 of the pairs correctly at 267 runs, against 0.749 for uniform allocation. Our 90% interval covers the official score in 0.87 to 0.91 of the cases, slightly below the nominal level at the larger budgets.

The check is cheap insurance rather than a free one: it costs about 0.03 points when the training fits, and always hedging instead would cost five to eight times that. Table C.18 reports the misfit check of Section 3.5 on five settings. On Terminal-Bench 2.0, where the training agents and the test agents come from the same release, the check changes the score error by about 0.03 percentage points, so the gain of deciding reruns survives. When the items are calibrated on Terminal-Bench 2.0 and the test agents come from 2.1, deciding reruns without the check is worse than uniform reruns at every budget above 134 runs, 1.30 against 1.15 points at 214 runs, and the check turns this into 1.01. We then confirmed the rule on Terminal-Bench 4.0, which we had not used to design it: the check is never significantly worse than uniform reruns, and significantly better at 99 and 264 runs.

Detection earns its place against always hedging, not against trusting the model. On Terminal-Bench 2.0, where the calibration does fit the test agents, spending half of every batch uniformly regardless costs 0.126 to 0.179 percentage points against the gated rule at six of the seven budgets, while the gated rule itself costs 0.016 to 0.029 points against trusting the model at two of the seven. The detection therefore buys the cross-release protection of Table C.18 at between a fifth and an eighth of the price of buying it unconditionally. The rule itself is calibrated: its threshold is the fifth percentile of the training reference, and it fires on 8.5% of test agents when the items come from their own release and on 56.7% when the items come from Terminal-Bench 2.0 and the agents from 2.1. The cost is paid on the first share and the protection is collected on the second.

The gain comes from deciding per agent, not from a fixed repetition count per task. A benchmark could instead fix how many attempts each task gets, once, for every agent. We allocated a total budget across tasks by the same expected variance reduction, using the training agents' item parameters, and compared it with equal counts at the same total. Over the 30 splits the two differ at only one of the six budgets: 1.70 against 1.86 percentage points of score error at 134 runs, and no significant difference at 89, 178, 267, 356 or 445 runs. Tasks differ enough in their interaction scale to make a per-agent decision pay, but not enough to make one fixed schedule better than equal counts.

The rerun savings do not depend on the prior on the interaction scale. The scale σ_j of every task carries a half-normal prior whose scale is 3 in the main experiment. Refitting with scales 1 and 10 and repeating the allocation on the same 30 splits leaves every number in place: our allocation's error is 1.34–1.36 points at 134 runs, 0.81 at 214, 0.54–0.57 at 267 and 0.20–0.21 at 356 under the three priors, against 1.72–1.75, 1.16–1.17, 0.84–0.91 and 0.53–0.55 for uniform reruns. With 95 training agents and at least five attempts per cell, the data, not the prior, set σ_j.

Charging runs by their cost changes which cells to rerun, and the allocation adapts. Tasks on Terminal-Bench 2.0 differ 34-fold in the tokens a run consumes (Appendix D.9), so a run of an expensive task should have to promise more. We divide each cell's expected variance reduction by its task's mean per-run token count and buy reruns in that order until a token budget is spent; the per-run rule and

Setting	Runs	Uniform reruns	Deciding reruns	With the check
TB 2.0, same release	134	1.79	1.34	**1.34**
	214	1.17	0.79	**0.80**
	356	0.56	0.21	**0.22**
TB 2.1, items from TB 2.0	134	1.76	1.54	**1.51**
	214	1.15	1.30	**1.01**
	356	0.48	0.61	**0.40**
TB 2.1, same release	134	1.58	1.49	1.43
	214	0.99	1.17	1.14
	356	0.47	0.43	**0.38**
TB 4.0, same release	99	2.18	1.92	**1.88**
	158	1.17	1.11	1.03
	264	0.64	0.29	**0.31**

Table C.18: **Score error when the item parameters may not fit the test agents**, in percentage points. With the check, an agent whose first pass is unlikely under the model falls back to spending half of each batch uniformly. Terminal-Bench 2.1 with items from 2.0 is the case where the release changed under the model. Bold: significantly below uniform reruns.

Runs per test agent (official protocol: 445)	89	134	178	214	267	356
Uniform reruns, cell mean	2.12	1.72†	1.34†	1.16†	0.87†	0.55†
Uniform reruns, model estimate	2.05	1.62†	1.33†	1.12†	0.87†	0.55†
Rerun mixed cells first	2.12	1.78†	1.49†	1.24†	0.99†	0.51†
Binomial-2PL allocation	2.17	1.64†	1.23†	0.95†	0.67†	0.24†
Ours	**2.05**	**1.36**	**1.01**	**0.81**	**0.54**	**0.20**
Kendall τ, ours / uniform	0.927 / 0.928	0.951 / 0.939	0.965 / 0.955	0.972 / 0.959	0.979 / 0.967	0.992 / 0.980
Close pairs ordered, ours / uniform	0.64 / 0.59	0.69 / 0.64	0.76 / 0.68	0.81 / 0.72	0.86 / 0.75	0.95 / 0.84
Coverage of our 90% interval	0.91	0.91	0.89	0.88	0.87	0.88

Table C.19: **Deciding reruns on Terminal-Bench 2.0.** Every method first runs all 89 tasks once and then spends the same number of further runs, each returning one archived attempt drawn without replacement. Upper block: mean absolute error of the official score, $\times 100$, over 30 splits; †: significantly above ours. Close pairs: test agents whose official scores differ by less than 0.03.

uniform reruns are charged the same costs and stopped at the same budgets (Table C.20). At every budget from one and a half to four first passes the per-token rule has lower error than the per-run rule, 1.23 against 1.35 points at 1.5 passes and 0.46 against 0.55 at 3, all four differences significant, and both are far below uniform reruns (1.75 and 0.92). Ranking does not pay for it: Kendall τ among the test agents is 0.950 against 0.949 at 1.5 passes and significantly higher at 3 and 4 passes (0.982 against 0.978, 0.994 against 0.990). The mechanism is quantity: the cells with the largest variance reduction tend to belong to expensive tasks, so at the same token budget the per-token rule buys many more reruns, on one split 85 against 43 at 1.5 passes and 139 against 88 at 2, and many slightly less valuable reruns beat fewer of the most valuable ones.

This is the opposite of what Appendix D.8 found for first-round selection, and the two results are consistent. Selecting the first tasks by information per dollar can skip an expensive task altogether, and on Terminal-Bench 2.1 the cheap tasks are also the ones that do not separate agents, so ranking fell from 0.78 to 0.70; the paper therefore does not adopt cost-aware selection. Cost-aware reruns start after every task has been run once, so no task can be skipped and cost only orders the reruns among cells that have all been observed; the failure mode of first-round selection cannot occur. The token costs here are per-task means over the nine submissions with released counts, charged to every agent alike, a coarser price than the per-agent dollar costs of Appendix D.8. The objective is the same; only the price of a run enters it, which is how the allocation should be used when costs are known.

C.2.4 The same posterior predicts pass@5 and pass^5

Leaderboards that run every task several times can report more than the mean: the share of tasks solved at least once in five attempts (pass@5) and the share solved every time (pass^5, the statistic τ-bench reports). Both are higher moments of the cell probability, $1 - \mathbb{E}[(1 - p_{ij})^5]$ and $\mathbb{E}[p_{ij}^5]$, so

Metric	Allocation	Token budget, in first passes of the bank 1.5	2	3	4
Score MAE	VOI per run (Section 3.5)	1.35	0.98	0.55	0.22
Score MAE	VOI per token	1.23*	0.91*	0.46*	0.17*
Score MAE	Uniform reruns	1.75†	1.33†	0.92†	0.57†
Kendall τ	VOI per run (Section 3.5)	0.949	0.963	0.978	0.990
Kendall τ	VOI per token	0.950	0.963	0.982*	0.994*
Kendall τ	Uniform reruns	0.937†	0.949†	0.965†	0.978†

Table C.20: **Allocating reruns per token instead of per run.** Terminal-Bench 2.0, 30 splits, the repeat-aware model throughout. Each task is charged its mean per-run token count; one first pass of the bank costs 55M tokens. Reruns are then bought in batches of eight by expected variance reduction per run, per token, or uniformly. †/*: significantly worse/better than the per-run rule.

Stopping rule	$\varepsilon = 3$ points runs	met	unif.	$\varepsilon = 2$ points runs	met	unif.	$\varepsilon = 1$ points runs	met	unif.
Reported, uncalibrated	131	0.90	169	183	0.89	249	266	0.84	356
Calibrated, not adopted	153	0.95	201	221	0.94	297	316	0.76	329

Table C.21: **Stopping by runs on Terminal-Bench 2.0, where a task can be run again.** Tolerances are one point tighter than in the task-counted table because reruns reach them. *unif.*: the runs uniform rerunning needs for the share to its left. The reported rule carries no replay margin and states no reliability.

they depend on the distribution of p_{ij} and not only on its mean. A model that treats attempts as independent replaces that distribution by a point, and Jensen's inequality fixes the direction of its error: $(1-p)^5$ is convex, so it overstates pass@5, and p^5 is convex, so it understates pass^5. Our agent–task random effect carries the distribution, and the acquired runs of a cell move it: after a failure the posterior on p_{ij} shifts down, after a success up.

Table C.22 scores the same acquired runs with three estimators of the archive's five-attempt statistics. Each estimator integrates the unobserved attempts of a cell over its posterior, ours over ability and the random effect, the binomial 2PL over ability alone, and the plug-in over the cell's observed rate. After one run of every task, the binomial model overstates pass@5 by 6.6 points and understates pass^5 by 6.6 on average, our estimator's mean signed error is $+0.9$ and -0.4 points, and the plug-in, which sees a single outcome per cell, is off by 16 and 17. These are biases. The mean absolute errors at the same budget are 2.86 and 3.21 points for our estimator against 6.73 and 6.88 for the binomial model, and Table C.22 reports both. The binomial error is still 2.1 and 2.5 points at three runs per task and vanishes only when the archive is nearly exhausted; under uniform reruns it remains 2.9 points at four runs per task. Under our allocation the three estimators are within 0.1 points of one another at 356 runs, because by then almost every uncertain cell has been resolved by observation. For the official mean the three estimators differ by a tenth of a point, as in Section 5.2: the random effect matters little for the mean and a great deal for pass@5 and pass^5, which is where a benchmark that reports them will need it.

C.3 Fixed budget: all budgets

Table C.23 gives Kendall τ at every budget when agents are ranked by the estimated ability rather than by the predicted score.

C.4 Score prediction at all budgets

Table C.24 gives the share of test agents a tenth of the bank keeps within each tolerance, Table C.25 compares every method at a tenth of the bank and Table C.26 at a fifth, and Table C.27 repeats the score comparison of Section C.1.1 at every budget, including the ones below four tasks that the main text omits.

Allocation	Target	Estimator	Runs per test agent 89	134	178	267	356
our allocation	pass@5	Ours	2.86 (+0.9)	2.39 (+0.8)	2.10 (+0.7)	1.41 (+0.1)	0.55 (+0.0)
our allocation	pass@5	Binomial 2PL	6.73 (+6.6)†	6.22 (+6.1)†	5.44 (+5.3)†	2.48 (+2.1)†	0.65 (+0.3)†
our allocation	pass@5	Plug-in	15.96 (-16.0)†	9.25 (-9.2)†	6.36 (-6.4)†	2.49 (-2.5)†	0.49 (-0.5)*
our allocation	pass^5	Ours	3.21 (-0.4)	2.76 (-0.4)	2.42 (-0.2)	1.53 (-0.1)	0.51 (+0.0)
our allocation	pass^5	Binomial 2PL	6.88 (-6.6)†	6.45 (-6.2)†	5.63 (-5.4)†	2.82 (-2.5)†	0.63 (-0.3)†
our allocation	pass^5	Plug-in	17.02 (+17.0)†	10.35 (+10.3)†	7.39 (+7.4)†	2.99 (+3.0)†	0.59 (+0.6)†
our allocation	mean (official)	Ours	2.07	1.37	1.01	0.57	0.21
our allocation	mean (official)	Binomial 2PL	2.18†	1.52†	1.13†	0.60†	0.23†
our allocation	mean (official)	Plug-in	2.09	1.65†	1.33†	0.71†	0.20*
uniform reruns	pass@5	Ours	2.86 (+0.9)	2.56 (+0.9)	2.23 (+0.8)	1.68 (+0.6)	1.08 (+0.3)
uniform reruns	pass@5	Binomial 2PL	6.73 (+6.6)†	6.32 (+6.2)†	5.95 (+5.9)†	4.76 (+4.7)†	2.99 (+2.9)†
uniform reruns	pass@5	Plug-in	15.96 (-16.0)†	11.89 (-11.9)†	7.96 (-8.0)†	4.10 (-4.1)†	1.72 (-1.7)†
uniform reruns	pass^5	Ours	3.21 (-0.4)	2.85 (-0.4)	2.50 (-0.4)	1.82 (-0.3)	1.23 (-0.2)
uniform reruns	pass^5	Binomial 2PL	6.88 (-6.6)†	6.48 (-6.3)†	6.11 (-6.0)†	4.89 (-4.8)†	2.99 (-2.9)†
uniform reruns	pass^5	Plug-in	17.02 (+17.0)†	12.95 (+13.0)†	8.92 (+8.9)†	4.68 (+4.7)†	2.02 (+2.0)†
uniform reruns	mean (official)	Ours	2.07	1.71	1.32	0.91	0.55
uniform reruns	mean (official)	Binomial 2PL	2.18†	1.83†	1.42†	0.98†	0.59†
uniform reruns	mean (official)	Plug-in	2.09	1.80†	1.34	0.89	0.55

Table C.22: **Predicting pass@5 and pass^5 on Terminal-Bench 2.0 from the same runs.** Every test agent runs each task once and then receives reruns in batches of eight, allocated by our model or uniformly. At each budget the same acquired runs are scored by three estimators of the same archive statistic. Entries: MAE in percentage points over 30 splits, signed bias in parentheses; †/*: significantly worse/better than ours.

Method	Verified 10%	20%	30%	50%	Lite 10%	20%	30%	50%	CORE 10%	20%	30%	50%	τ-bench 10%	20%	30%	50%
Random	0.848†	0.899†	0.922†	0.951†	0.756†	0.820†	0.858†	0.901†	0.428†	0.631†	0.730†	0.812†	0.403†	0.535†	0.624†	0.717†
Random + IRT	0.857†	0.901†	0.920†	0.943†	0.743†	0.805†	0.839†	0.885†	0.437†	0.663†	0.748†	0.811†	0.403†	0.482†	0.519†	0.565†
FLUID	0.895†	0.913†	0.928†	0.945†	**0.853**	0.888†	0.907†	0.930†	0.585†	0.752	0.814	0.871†	0.453†	0.513†	0.556†	0.600†
ATLAS	0.883†	0.906†	0.920†	0.936†	0.838	0.874†	0.896†	0.911†	0.594†	0.738	0.790†	0.848†	0.447†	0.529†	0.557†	0.569†
Ours	**0.901**	**0.925**	**0.938**	**0.962**	0.853	**0.927**	**0.944**	**0.968**	**0.652**	**0.762**	**0.826**	**0.889**	0.515	**0.637**	**0.734**	**0.823**

Table C.23: **Kendall τ at a fixed budget**, as a share of the bank, on the single-attempt benchmarks; mean over 30 splits. Rankings are by estimated ability, as in the baselines' papers. Bold: best; underline: second. †/*: significantly below/above ours.

C.5 Additional operating-point notes

The calibrated certificate cannot support arbitrarily small stated miss rates with 30 internal splits because its high empirical quantiles equal the maximum observed error. The multiplicative certificate avoids a quantile ceiling by increasing K with safety factor c, but it trades a probabilistic interpretation for an empirical tuning knob. Both variants are prospective: neither observes a test agent before fixing its budget.

D Component Studies

Each part below isolates one decision. The first five take the components C1–C7 in turn: item calibration, the selection rule, the ability estimate, the stopping rule and the treatment of repeats. The remainder address questions those raised: where a run budget should go, what a matched reference shows, what an operator actually pays, which designs we tried and dropped, and what survived on the benchmarks we did not design on.

Bank (tasks)	tasks at 10%	$\varepsilon = 4$ points		$\varepsilon = 3$ points		$\varepsilon = 2$ points	
		best baseline	ours	best baseline	ours	best baseline	ours
Verified (500)	50	0.81	0.83	0.68	0.71	0.53	0.54
Lite (300)	30	0.73	0.80	0.59	0.68	0.43	0.52
CORE (45)	4	0.36	0.39	0.28	0.29	0.19	0.19
τ-bench (50)	5	0.30	0.40	0.23	0.32	0.17	0.23
TB 2.0 (89)	9	0.55	0.53	0.43	0.41	0.30	0.27

Table C.24: **A tenth of the bank is a cost, not a reliability.** Share of test agents within ε of their full-bank score at that budget. Mean over 30 splits.

Method	Verified (50)		Lite (30)		CORE (4)		τ-bench (5)	
	MAE	τ	MAE	τ	MAE	τ	MAE	τ
tinyBenchmarks	3.97†	0.865†	4.66†	0.782†	14.38†	0.465†	15.43†	0.312†
CTT panel	3.27†	0.889†	3.96†	0.820†	7.88†	0.544†	8.00	0.465
Mid-range panel	4.49†	0.882†	3.44†	0.850	8.46†	0.565†	8.17	0.513
PTA-IRT	3.08†	0.876†	3.90†	0.794†	11.89†	0.559†	7.80	**0.527**
FLUID	2.47†	0.897	2.83†	**0.857**	8.81†	0.587†	8.25†	0.458†
ATLAS	2.44	0.901	3.06†	0.845	7.55†	**0.651**	9.01†	0.490
TITRATE	**2.34**	**0.902**	**2.38**	0.855	**6.91**	0.649	**7.58**	0.509

Table C.25: **Score error and ranking quality at a tenth of the bank**, tasks in parentheses. MAE in percentage points; τ: Kendall correlation among a split's held-out agents. Mean over 30 splits; bold: best in the column; †/*: significantly worse/better than TITRATE.

D.1 C1, C2: ITEM CALIBRATION DECIDES ALMOST EVERYTHING

Item calibration explains the rest. Table D.1 swaps one component at a time from a baseline into our method. Each swapped component comes from the baseline's released code, fitted on the same split. Replacing both FLUID's item parameters and its estimator reproduces FLUID's official results to within 0.004, which validates the harness. The estimator does not matter: FLUID's MAP estimator changes our τ by at most 0.006 in every cell. The item parameters matter. With FLUID's parameters our method loses 0.012 on Terminal-Bench 2.0 at 20%, 0.056 on CORE-bench at 10% and 0.101 on τ-bench at 20%, which is most of its advantage over FLUID in each cell where it is ahead. With ATLAS's parameters it loses 0.012 to 0.050. ATLAS's randomesque selection, which picks at random among the five most informative tasks, costs 0.047 on CORE-bench at 10% and little elsewhere.

The model form does not matter; the estimator does. ATLAS uses a 3PL model and we use a 2PL, so the comparison above leaves open whether the form is what separates them. We fitted 2PL, 3PL and 4PL with the same estimator, the same shrinkage on discriminations, and the same selection rule, shrinking the guessing parameter toward zero and the upper asymptote toward one so that a task stays at the boundary unless the data move it. Between 1 and 21 tasks per bank do move, most of them on SWE-bench Verified, but ranking quality and score error agree to within 0.011 everywhere, and to the third decimal on the three largest banks, and choosing the form per split by internal validation does not help. Over the 18 comparisons the 3PL is significantly better in 1 and worse in 2, and the 4PL better in 1 and worse in none. For these benchmarks a 2PL is enough, and the gap to the baselines comes from how the parameters are estimated, not from how many there are.

Why the item parameters differ. With 35 to 130 training agents, a task can separate the training agents well by chance and receive an inflated discrimination. Maximum-information selection then prefers exactly these tasks, and they carry less information on test agents than the model assumes. This is capitalization on calibration error, known in adaptive testing (van der Linden & Glas, 2000). It is mild when items are calibrated on many examinees, as in the language-model setting of FLUID (102 models) and ATLAS (thousands), and strong on agentic benchmarks. The fitted discriminations show it. Over the first ten splits of Terminal-Bench 2.0, the 95th percentile of discrimination is 1.81 for our shrunken fit, 3.12 for FLUID's fit and 33.3 for ATLAS's three-parameter fit. On τ-bench airline, with 35 training agents, the values are 1.23, 2.72 and 55.3. Our penalty toward $a_j = 1$ removes these

	Verified (100)			Lite (60)			CORE (9)			τ-bench (10)		
Method	MAE	τ	ρ	MAE	τ	ρ	MAE	τ	ρ	MAE	τ	ρ
tinyBenchmarks	3.13†	0.908†	0.984†	3.09†	0.857†	0.954†	8.89†	0.622†	0.778†	9.55†	0.525†	0.663†
CTT panel	2.70†	0.915†	0.986	3.10†	0.868†	0.965†	5.78†	0.692†	0.860†	6.29	0.595	0.763
Mid-range panel	4.08†	0.914†	0.987	2.62†	0.898†	0.975†	6.36†	0.726†	0.883	6.46	**0.653**	**0.811**
PTA-IRT	2.47†	0.905†	0.983†	3.09†	0.849†	0.955†	8.51†	0.665†	0.819†	7.26†	0.590	0.741
FLUID	1.89†	0.918†	0.986†	2.05†	0.887†	0.967†	5.88†	0.751	0.881	7.00†	0.538†	0.675†
ATLAS	1.89	0.924	0.987	2.23†	0.885†	0.967†	5.43	0.757	**0.890**	7.91†	0.547†	0.689†
TITRATE	**1.77**	**0.925**	**0.987**	**1.56**	**0.923**	**0.982**	**5.09**	**0.763**	0.890	**6.13**	0.637	0.777

Table C.26: **The same comparison at a fifth of the bank, with Spearman ρ.** Marks as above.

Benchmark	Tasks run	Ours	Random
Verified	25	2.97	7.77†
Verified	50	2.34	5.65†
Verified	100	1.77	3.37†
Verified	150	1.43	2.41†
Verified	250	0.84	1.76†
Lite	15	3.42	7.93†
Lite	30	2.38	5.82†
Lite	60	1.56	4.46†
Lite	90	1.11	3.59†
Lite	150	0.67	2.29†
CORE	2 (below floor)	8.84	22.86†
CORE	4	6.91	15.57†
CORE	9	5.09	9.08†
CORE	14	3.56	6.57†
CORE	22	2.29	4.42†
τ-bench	2 (below floor)	9.76	28.49†
τ-bench	5	7.58	15.14†
τ-bench	10	6.13	10.30†
τ-bench	15	4.86	8.41†
τ-bench	25	3.28	5.68†

Table C.27: **Error of the predicted full-bank pass rate**, percentage points, at every budget from a twentieth to half the bank, including budgets below the four-task floor of the main text. †/*: significantly worse/better than ours.

outliers. This is the reason our method is better within the same family: it is designed for few training agents.

The shrinkage strength should be read off the training pool, not fixed. A fixed penalty is not enough on its own. With λ fixed at 10, our expected calibration error is worse than FLUID's on SWE-bench Verified, the largest pool. Choosing λ from a grid by internal validation on the training agents alone removes both: the selected value is 1 to 3 on the pools with 63 to 130 agents and 30 to 100 on the pools with 9 to 15, and our calibration error becomes lower than FLUID's on SWE-bench Verified (0.010 against 0.012) while the other metrics do not move. The selection costs three extra fits per split and needs no new data.

The advantage does not depend on the penalty strength. Table D.2 varies the penalty λ from 0 to 100, with and without the cap on discrimination. Without either, our method only ties FLUID, and on CORE-bench Hard it is worse. Any limit on extreme discriminations restores the advantage. With the cap and any λ from 0 to 100, our method is never significantly below FLUID and is significantly above it on four of the five benchmarks. The sweep holds λ fixed, with 10 as the reference value; it is not the best value, since stronger penalties do better on three benchmarks, and the deployed method chooses λ by the internal rule of Section D.13 instead.

Swapped into our method	TB 2.0 9	TB 2.0 18	Verified 50	Verified 100	Lite 30	Lite 60	CORE 4	CORE 9	τ-bench 5	τ-bench 10
Ours (τ)	0.858	0.909	0.899	0.925	0.870	0.910	0.660	0.775	0.484	0.617
FLUID's item parameters	$+0.003$	$-0.012^{\dagger}$	$-0.004^{\dagger}$	$-0.011^{\dagger}$	$-0.014^{\dagger}$	$-0.019^{\dagger}$	$-0.056^{\dagger}$	-0.016	-0.031	$-0.101^{\dagger}$
ATLAS's 3PL parameters	$-0.012^{\dagger}$	$-0.021^{\dagger}$	$-0.016^{\dagger}$	$-0.018^{\dagger}$	$-0.039^{\dagger}$	$-0.036^{\dagger}$	$-0.050^{\dagger}$	$-0.045^{\dagger}$	-0.019	$-0.083^{\dagger}$
FLUID's MAP estimator	$+0.001$	-0.002	$+0.001$	-0.000	-0.001	-0.002	-0.002	$+0.001$	$+0.004$	-0.006
ATLAS's randomesque selection	$+0.004$	-0.006	$+0.001$	-0.001	-0.006	-0.000	$-0.047^{\dagger}$	-0.001	$+0.026$	$+0.008$
FLUID's parameters and estimator	-0.000	$-0.016^{\dagger}$	-0.004	$-0.012^{\dagger}$	$-0.017^{\dagger}$	$-0.021^{\dagger}$	$-0.071^{\dagger}$	-0.022	-0.033	$-0.105^{\dagger}$

Table D.1: **Which component makes the difference.** One component of our method is replaced by the one a baseline uses, fitted on the same split. Headers give the benchmark and the number of tasks. First row: our Kendall τ; other rows: the change in τ, mean over 30 splits. †: significantly lower than ours.

Discrimination penalty	Verified 100	Lite 60	CORE 9	τ-bench 10	TB 2.0 18
FLUID	0.913	0.888	0.752	0.513	0.893
λ=0, no cap	0.917^{+}	0.887	0.716^{-}	0.508	0.896
λ=0	0.922^{+}	0.897^{+}	0.764	0.550^{+}	0.905^{+}
λ=1	0.922^{+}	0.901^{+}	0.772	0.567^{+}	0.906^{+}
λ=3	0.921^{+}	0.908^{+}	0.780	0.593^{+}	0.909^{+}
λ=10 (default)	0.925^{+}	0.910^{+}	0.775	0.617^{+}	0.909^{+}
λ=30	0.928^{+}	0.913^{+}	0.779	0.637^{+}	0.909^{+}
λ=100	0.930^{+}	0.928^{+}	0.753	0.661^{+}	0.908^{+}

Table D.2: **Sensitivity to the strength λ of the penalty that pulls discriminations toward 1**, at a fifth of the bank, task counts after the benchmark name. Discriminations are capped to $[0.4, 2.5]$ except in the first row. Kendall τ, mean over 30 splits; $^{+}$/$^{-}$: significantly above/below FLUID.

The bounds on discrimination matter little once λ is chosen by internal validation. Table D.3 reruns the deployed single-attempt pipeline with the bounds $[0.25, 4.0]$ and $[0.5, 2.0]$ and with no cap, next to the default $[0.4, 2.5]$, and re-chooses λ under each. Over the ten benchmark–budget cells, no alternative moves Kendall τ by more than 0.021 or the score error by more than 0.29 points. Each alternative has 20 paired comparisons with the default. Each is significantly better in three of them and significantly worse in one to three, with no consistent direction. The wider bounds and no cap agree to three decimals on SWE-bench Verified and Lite, where at most 1% of discriminations reach either bound. In Table D.2 the cap was removed together with the penalty. With the penalty chosen by validation, the cap is no longer what limits extreme discriminations. Our guidance is therefore to choose λ by internal validation on the training agents and to treat the bounds as a guard against numerical extremes, not as a parameter to tune. Any of the bounds we tested serves.

Shrinkage keeps the discriminations rather than erasing them. A Rasch model answers whether the cap and penalty simply pull the 2PL back to one-parameter behavior: we fix every a_j at 1, fit only difficulties and abilities, and leave selection and scoring unchanged, so that maximum-information selection reduces to the task whose difficulty is closest to the current ability estimate (Table D.4). The 1PL is significantly worse in 8 of the 10 cells, by 0.50 points on SWE-bench Verified at a tenth, 1.22 on CORE-bench Hard, 1.02 on τ-bench airline and 1.07 on Terminal-Bench 2.0, and never significantly better; the remaining cells are ties at a fifth of the bank. With a few dozen training agents the discriminations therefore still carry information about which tasks separate agents; shrinkage limits the noise in that information without discarding it.

The discrimination is not reproducible from the pool that estimates it. Our explanation so far has been that selection capitalises on noisy discriminations. That can be measured directly. Cutting each split's training pool in half at random and fitting both halves without shrinkage, the two estimates of the difficulty correlate at 0.89, 0.82, 0.79, 0.75 and 0.27 across the five benchmarks, while the two estimates of the discrimination correlate at only 0.66, 0.32, 0.22, 0.50 and 0.20, and the correlation falls with the pool. A pool of 17 to 47 agents per half cannot reproduce its own discriminations. Whether this is what shrinking corrects is measured in Section D.13: at a tenth of SWE-bench Verified a pool whose discriminations reproduce better wants less shrinkage, and at a half of it the relation reverses. The link holds at the budgets where selection is doing the work, and not at the budgets where little is left to select.

Bounds on a_j	Verified 50/100		Lite 30/60		CORE 4/9		τ-bench 5/10		TB 2.0 9/18	
	τ	err	τ	err	τ	err	τ	err	τ	err
A tenth of the bank										
$[0.25, 4.0]$	0.900	2.32	0.862	2.40^{-}	0.645	7.18	0.486	7.92	0.866^{+}	3.98^{+}
$[0.4, 2.5]$ (default)	0.901	2.28	0.864	2.38	0.646	7.05	0.494	7.87	0.855	4.19
$[0.5, 2.0]$	0.902	2.27	0.867	2.31	0.630	7.24	0.490	7.90	0.857	4.16
no cap	0.900	2.32	0.862	2.40^{-}	0.655	7.05	0.482^{-}	7.84	0.864^{+}	3.99
A fifth of the bank										
$[0.25, 4.0]$	0.921	1.82^{-}	0.912	1.61	0.762	5.11	0.619^{+}	6.45	0.906	3.00
$[0.4, 2.5]$ (default)	0.922	1.76	0.912	1.61	0.774	4.98	0.607	6.48	0.906	2.94
$[0.5, 2.0]$	0.925^{+}	1.77	0.909	1.60	0.753^{-}	5.05	0.624	6.19^{+}	0.911	2.78^{+}
no cap	0.921	1.82^{-}	0.912	1.61	0.769	4.98	0.625^{+}	6.32^{+}	0.906	3.01

Table D.3: **Sensitivity to the bounds on discrimination.** The single-attempt pipeline is rerun with each pair of bounds; λ is re-chosen by the internal log-loss rule under the same bounds, so each row is what an operator would deploy with those bounds. "No cap" keeps only the numerical guard $[0.05, 20]$. Task counts at a tenth and a fifth after the benchmark name. τ: Kendall correlation with the full-bank order; err: score error in percentage points; means over the 30 random splits. $^{+}/^{-}$: significantly better/worse than the default bounds (paired 95% bootstrap interval over splits).

Bank (tasks)	10% of the bank						20% of the bank					
	Ours (2PL, Fisher)		1PL, Fisher		2PL, score-variance VOI		Ours (2PL, Fisher)		1PL, Fisher		2PL, score-variance VOI	
	MAE	τ	MAE	τ	MAE	τ	MAE	τ	MAE	τ	MAE	τ
SWE-bench Verified (500)	2.34	0.902	$2.84^{\dagger}$ (+0.50)	0.901	2.32 (-0.02)	0.901	1.77	0.925	$1.93^{\dagger}$ (+0.16)	0.936	1.70^{*} (-0.07)	0.930
SWE-bench Lite (300)	2.38	0.855	$2.58^{\dagger}$ (+0.21)	0.856	2.43 (+0.05)	0.853	1.56	0.923	1.54 (-0.02)	0.921	1.51 (-0.05)	0.924
CORE-bench Hard (45)	6.91	0.652	$8.13^{\dagger}$ (+1.22)	0.570	6.81 (-0.10)	0.637	5.09	0.763	$5.75^{\dagger}$ (+0.66)	0.700	4.85^{*} (-0.24)	0.774
τ-bench airline (50)	7.58	0.508	$8.60^{\dagger}$ (+1.02)	0.500	7.68 (+0.10)	0.521	6.13	0.637	6.39 (+0.27)	0.662	5.94 (-0.19)	0.662
SWE-bench Pro (730)	2.86	0.889	3.15 (+0.29)	0.922	2.79 (-0.06)	0.878	2.27	0.922	2.32 (+0.05)	0.922	2.37 (+0.10)	0.889
Terminal-Bench 2.0 (89)	4.43	0.857	$5.50^{\dagger}$ (+1.07)	0.824	4.50 (+0.08)	0.861	3.01	0.908	3.15 (+0.14)	0.901	2.95 (-0.06)	0.909

Table D.4: **Two ablations of the single-attempt procedure**, changing only the named component. 1PL: every discrimination fixed at 1. Score-variance VOI: select by the expected reduction of score-error variance rather than by Fisher information about ability. MAE in percentage points, τ Kendall, mean over 30 splits; parentheses give the paired MAE difference from ours; $^{\dagger}/^{*}$: significantly worse/better than ours.

D.2 C3, C4: THE SELECTION RULE AND THE ABILITY ESTIMATE ARE FREE, AND A SHARED PANEL IS NOT

Three alternatives to maximum-information selection change nothing or hurt. Sharing the first four tasks across all new agents, so that comparisons rest on common evidence, costs 0.020 to 0.069 Kendall τ at four to nine tasks and is significantly worse on six of the seven settings. Using one designed panel for every agent, the anchor-point design of tinyBenchmarks, costs 0.03 to 0.20. Weighting information by the current posterior instead of evaluating it at the point estimate changes τ by less than 0.01 everywhere. Replacing our grid posterior mean by FLUID's maximum a posteriori estimate moves τ by at most 0.006 (Table D.1). On these benchmarks, adaptivity itself matters; its details do not. Table D.5 gives the numbers.

The selection objective is not the evaluation objective, and changing it changes little. Maximum-information selection maximizes information about ability, while the reported error is that of the score, whose sensitivity to ability is $\sum_j a_j p_j (1 - p_j)$ without the square, and running a task also removes that task's own Bernoulli noise from the estimate, a term absent from Fisher information. The rerun stage already selects by the expected reduction of the score-error variance, so we ran the same criterion in the first round: for each unrun task, the posterior variance of the predicted score plus the Bernoulli noise of the tasks not yet run, before and after a hypothetical outcome, averaged over that outcome (Table D.4). The score-variance rule is never significantly worse and is significantly better in two of twelve cells, by 0.07 points on SWE-bench Verified and 0.24 on CORE-bench Hard at a fifth of the bank; elsewhere the two rules are within 0.1 points of each other. The main text keeps the information rule because it is the simpler one and the two agree; the score-variance rule remains the natural choice once tasks and reruns compete for the same budget.

	Maximum information		Four shared anchors		One designed panel		Posterior-weighted	
Benchmark	4–9	10%	4–9	10%	4–9	10%	4–9	10%
Verified	0.780	0.899	$-0.032^{\dagger}$	$+0.003$	$-0.082^{\dagger}$	$-0.047^{\dagger}$	$-0.007^{\dagger}$	$+0.001$
Lite	0.724	0.870	-0.006	-0.001	-0.009	$-0.029^{\dagger}$	-0.004	$+0.001$
CORE	0.731	0.658	$-0.020^{\dagger}$	$-0.042^{\dagger}$	$-0.033^{\dagger}$	$-0.042^{\dagger}$	-0.003	$-0.014^{\dagger}$
τ-bench	0.528	0.482	-0.020	-0.022	$-0.034^{\dagger}$	-0.024	-0.009	-0.005
TB 2.0	0.831	0.858	$-0.039^{\dagger}$	-0.007	$-0.099^{\dagger}$	$-0.088^{\dagger}$	-0.002	$+0.001$
Verified (time)	0.636	0.851	$-0.044^{\dagger}$	$+0.005$	$-0.113^{\dagger}$	$-0.018^{\dagger}$	-0.006	$+0.006$
Lite (time)	0.540	0.712	$-0.069^{\dagger}$	$+0.011$	$-0.150^{\dagger}$	$-0.110^{\dagger}$	-0.003	$+0.005$
TB 2.0 (time)	0.619	0.693	$-0.047^{\dagger}$	$-0.040^{\dagger}$	$-0.164^{\dagger}$	$-0.162^{\dagger}$	-0.001	-0.002

Table D.5: **Alternatives to maximum-information selection**, holding everything else fixed. First pair of columns: Kendall τ of our selection, averaged over budgets of four to nine tasks and at a tenth of the bank. Other columns: the change from it. †: significantly worse.

		Kendall τ on held-out bank			TITRATE − baseline items		Items
Held-out bank (tasks)	Selected by other four	selected	TITRATE	TITRATE − selected	FLUID	ATLAS	selected
Verified (50)	MAP ability	0.900	0.899	-0.001 [-0.003, +0.000]	+0.004 [+0.000, +0.009]	+0.016 [+0.011, +0.022]	yes
Lite (30)	MAP ability	0.869	0.870	+0.001 [-0.002, +0.003]	+0.014 [+0.001, +0.027]	+0.039 [+0.022, +0.058]	yes
CORE (4)	randomesque selection	0.613	0.660	+0.047 [-0.000, +0.091]	+0.056 [+0.014, +0.099]	+0.050 [+0.009, +0.090]	yes
τ-bench (5)	MAP ability	0.489	0.484	-0.004 [-0.017, +0.008]	+0.031 [-0.009, +0.070]	+0.006 [-0.065, +0.077]	yes
TB 2.0 (9)	MAP ability	0.860	0.858	-0.001 [-0.006, +0.004]	-0.003 [-0.011, +0.005]	+0.012 [+0.003, +0.020]	yes
Verified (100)	randomesque selection	0.924	0.925	+0.001 [-0.001, +0.003]	+0.011 [+0.007, +0.014]	+0.018 [+0.013, +0.024]	yes
Lite (60)	randomesque selection	0.910	0.910	+0.000 [-0.007, +0.007]	+0.019 [+0.007, +0.031]	+0.036 [+0.021, +0.052]	yes
CORE (9)	randomesque selection	0.774	0.775	+0.001 [-0.026, +0.025]	+0.016 [-0.006, +0.040]	+0.045 [+0.019, +0.070]	yes
τ-bench (10)	TITRATE	0.617	0.617	+0.000 [+0.000, +0.000]	+0.101 [+0.062, +0.143]	+0.078 [+0.009, +0.142]	yes
TB 2.0 (18)	randomesque selection	0.903	0.909	+0.006 [+0.000, +0.014]	+0.012 [+0.005, +0.019]	+0.021 [+0.015, +0.027]	yes

Table D.6: **Leave-one-benchmark-out selection of the components.** For each held-out bank the configuration with the highest mean Kendall τ over the other four is chosen and evaluated on the held-out one, beside what we report there; the first block is a tenth of the bank, the second a fifth. Differences are paired over 30 splits with 95% bootstrap intervals.

Under a time shift the shared panel loses much more. The alternatives above were measured on random agent splits. Repeating them across the six chronological cutoffs, maximum-information selection stays ahead of anchor items and of posterior-integrated information by the same margins, but the fixed shared panel falls from 0.02 to 0.05 behind on random splits to 0.11 on SWE-bench Lite and 0.16 on Terminal-Bench 2.0 at a tenth of the bank. A panel chosen on today's agents is a commitment, and the cohort moves away from it; choosing each agent's tasks at administration time is not.

D.3 IS THE CONFIGURATION CHOSEN ON THE BENCHMARKS IT IS EVALUATED ON?

The component study of Appendix D.1 swaps one part of TITRATE for a baseline's on the same splits. A reader may ask whether the configuration we report was selected by looking at those very results, on the very benchmarks the paper evaluates it on. Table D.6 answers this by leave-one-benchmark-out selection: for each bank, the configuration with the highest mean Kendall τ over the other four banks is chosen without looking at the held-out one, and then evaluated there.

Two things follow. First, the decisive component is selected under every hold-out. Our shrunk item parameters beat both FLUID's and ATLAS's on every four-bank selection set at both budgets, and on the held-out bank they lead FLUID's by $+0.004$ to $+0.101$ and ATLAS's by $+0.006$ to $+0.078$, significantly in 14 of the 20 paired comparisons and never significantly behind. Second, the two components that the study found immaterial, the ability estimator and the selection rule, are interchangeable, and the leave-one-out selection lands on one of them in eight of ten hold-outs at no measurable cost: the selected configuration and TITRATE differ by at most 0.006 in τ on the held-out bank in nine cells, with every interval covering zero. The exception is CORE-bench Hard at a tenth of the bank, four tasks, where the selection lands on the randomesque rule and loses 0.047 with an interval touching zero. The configuration we report is therefore not a product of tuning on the evaluation benchmarks: the choice that matters is unanimous, and the choices that do not matter do not move the numbers.

Benchmark	ϵ	Tasks used by the stopping rule			Target met		Signed error	
		mean	s.d.	10–90%	stop	fixed	stop	fixed
SWE-bench Lite	4	19	3	16–21	0.69	0.72	+0.33	+0.15
	3	31	5	27–34	0.63	0.69	+0.28	+0.05
	2	60	11	53–67	0.67	0.70	+0.15	+0.00
CORE-bench Hard	4	20	4	15–24	0.78	0.78	+0.01	−0.14
	3	25	4	20–30	0.84	0.79	−0.17	−0.15
	2	32	3	28–36	0.91	0.84	−0.25	−0.19
τ-bench airline	4	31	2	29–32	0.74	0.76	−0.50	−0.54
	3	36	2	34–37	0.75	0.76	−0.35	−0.34
	2	41	1	40–42	0.78	0.77	−0.13	−0.07
Terminal-Bench 2.0	4	15	5	9–21	0.66	0.68	+0.32	+0.22
	3	30	10	15–41	0.67	0.66	+0.32	+0.21
	2	71	21	42–87	0.62	0.63	+0.15	+0.11

Table D.7: **A per-agent stopping rule against a count fixed in advance at the same average cost.** ϵ: the target error of the predicted full-bank pass rate, in percentage points. Target met: the share of test agents whose realized error is within ϵ. Signed error: the mean error of the estimate, in percentage points.

D.4 C6: WHAT THE CALIBRATION OF THE STOPPING MARGIN BUYS

An uncalibrated rule is the natural first attempt: after every task, add the margin of Section 3.6 to the posterior standard deviation of the predicted full-bank pass rate and stop when the sum falls below ε. That margin is estimated from the *average* error of the training agents and then applied to a single agent, and it does not deliver the stated level: the share of test agents within the target is 0.62 to 0.91 against a stated 0.95, and the realized error of the stopped estimate is biased upward, by 0.33 percentage points on SWE-bench Lite against 0.15 for a fixed count, which is what a rule that stops when the model looks confident would produce, although we did not isolate that cause. Against a fixed count at the same average cost it is significantly better in 2 of 12 cells, worse in 4, and tied in the rest (Table D.7).

Calibrating the margin per agent is what closes the gap, without making the stated level a guarantee. On the internal splits we collect, for every training agent and every budget, the predicted error s and the realized error e, bin by s, take the 95th percentile of e/s within the bin, and stop when the calibrated value falls below the target; we also subtract the mean signed error that internal agents show at their own stopping time. The measured per-agent reliability is then at or above 0.90 in 10 of the 12 cells, against 1 of 12 before, and the bias falls from 0.25 to 0.06 percentage points. It is not cheaper: it buys that reliability by running 1.2 to 3 times more tasks, and at that cost a fixed count reaches the same reliability. What it adds is that the count follows from the tolerance rather than from a number the evaluator must already know (Table 3). Figure D.1 shows the mechanism: the model's own error estimate tracks the mean realized error, and only the calibrated criterion tracks the tail a 95% statement has to bound.

The run-level rule of Table C.21 stops on the posterior standard deviation without a margin and realizes 0.90, 0.89 and 0.84 at 3, 2 and 1 points against the nominal 0.95 of its uncalibrated posterior interval. Calibrating its margin the same way raises the realized share to 0.95 and 0.94 at 3 and 2 points, at 153 and 221 runs instead of 131 and 183, still 24–25% fewer runs than uniform reruns need for the same share; at 1 point the margin pushes most agents to the 356-run cap without stopping, the share falls to 0.76 against 0.92 for a count fixed at the same mean, and the pre-registered criterion fails, so we keep the uncalibrated rule.

Table D.8 sets the calibrated rule against every baseline's best possible stopping point. Since no baseline states when to stop, each is given the smallest budget at which its own mean curve already satisfies the criterion, read in hindsight, so the comparison favours the baselines. The hit criterion, 90% of agents within ε, is adopted for this comparison and is looser than the 0.95 the rule states. Under it we count only the twelve rows where our own rule meets that criterion; Verified at two points and Terminal-Bench 2.0 at three and two, where it does not, are shown but not counted. In 33 of the

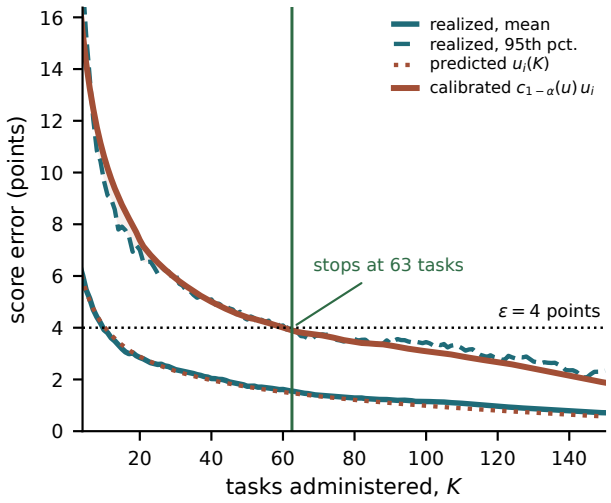

Figure D.1: **What the stopping rule sees, on SWE-bench Lite.** Mean over the test agents of 30 splits. The model's own predicted error follows the *mean* realized error, so it bounds a tail badly; after calibration on replayed training agents the criterion follows the 95th percentile instead and crosses a four-point tolerance at 63 tasks.

		Ours, prospective	Hindsight-optimal stopping point of each method													
Bank	ε	tasks (met)	Random		tinyBenchmarks		CTT panel		PTA-IRT		FLUID		ATLAS		Ours	
			hit	MAE	hit	MAE	hit	MAE	hit	MAE	hit	MAE	hit	MAE	hit	MAE
Verified	4	112 (0.94)	224	86	n.r. (>100)	≤50	211	27	245	≤25	89	13	103	≤25	87	13
	3	170 (0.93)	n.r. (>250)	119	n.r. (>100)	n.r. (>100)	318	71	n.r. (>250)	57	172	27	184	29	163	24
	2	191 (0.81)	n.r. (>250)	213	n.r. (>100)	n.r. (>100)	410	216	n.r. (>250)	196	260	86	n.r. (>250)	90	240	75
Lite	4	63 (0.94)	n.r. (>150)	76	n.r. (>60)	43	164	29	n.r. (>150)	29	68	14	85	≤15	44	10
	3	105 (0.94)	n.r. (>150)	117	n.r. (>60)	n.r. (>60)	205	65	n.r. (>150)	67	113	27	138	32	72	20
	2	148 (0.94)	n.r. (>150)	n.r. (>150)	n.r. (>60)	n.r. (>60)	249	144	n.r. (>150)	n.r. (>150)	158	63	n.r. (>150)	73	115	40
CORE	4	30 (0.95)	n.r. (>22)	n.r. (>22)	n.r. (>9)	n.r. (>9)	35	21	n.r. (>22)	n.r. (>22)	29	16	n.r. (>22)	16	28	12
	3	33 (0.95)	n.r. (>22)	n.r. (>22)	n.r. (>9)	n.r. (>9)	40	29	n.r. (>22)	n.r. (>22)	33	21	n.r. (>22)	n.r. (>22)	31	17
	2	38 (0.97)	n.r. (>22)	n.r. (>22)	n.r. (>9)	n.r. (>9)	42	36	n.r. (>22)	n.r. (>22)	38	28	n.r. (>22)	n.r. (>22)	34	24
τ-bench	4	41 (0.97)	n.r. (>25)	n.r. (>25)	n.r. (>10)	n.r. (>10)	44	25	n.r. (>25)	n.r. (>25)	43	30	n.r. (>25)	n.r. (>25)	38	20
	3	44 (0.97)	n.r. (>25)	n.r. (>25)	n.r. (>10)	n.r. (>10)	45	35	n.r. (>25)	n.r. (>25)	46	37	n.r. (>25)	n.r. (>25)	42	27
	2	45 (0.98)	n.r. (>25)	n.r. (>25)	n.r. (>10)	n.r. (>10)	n.r. (>48)	42	n.r. (>25)	n.r. (>25)	48	43	n.r. (>25)	n.r. (>25)	43	37
TB 2.0	4	64 (0.91)	n.r. (>44)	n.r. (>44)	n.r. (>18)	n.r. (>18)	70	25	n.r. (>44)	25	51	11	n.r. (>44)	11	50	11
	3	83 (0.83)	n.r. (>44)	n.r. (>44)	n.r. (>18)	n.r. (>18)	n.r. (>85)	38	n.r. (>44)	n.r. (>44)	n.r.	18	n.r. (>44)	19	n.r.	18
	2	89 (0.62)	n.r. (>44)	n.r. (>44)	n.r. (>18)	n.r. (>18)	n.r. (>85)	66	n.r. (>44)	n.r. (>44)	n.r.	43	n.r. (>44)	43	n.r.	46

Table D.8: **Our stopping rule against every baseline's best possible stopping point.** Ours: the mean count at which the calibrated rule stops, and the share within ε; the rule and its margin are fixed before the test agent runs, and the count depends on its outcomes. Every other entry is read off the method's own curve in hindsight, so it is a lower bound on that method's cost: the smallest budget at which 90% of agents are within ε (hit, a criterion adopted for this comparison and looser than the 0.95 the rule states) or the mean error is below ε (MAE). $\leq K$: met at the smallest budget evaluated; n.r.: not met at the largest.

remaining 72 cells the baseline has not met the criterion at the last budget its released code was run at, and that budget lies below our count, so neither method can be called the cheaper; these cells carry the grid's end. In the 39 cells where both counts are determined, our prospective count is at or below the hindsight bound in 33. The six exceptions are Verified at four points, where FLUID's and ATLAS's bounds are 89 and 103 tasks against our 112, Terminal-Bench 2.0 at four points, where FLUID's is 51 against our 64, and CORE-bench Hard against FLUID at all three targets, where the bounds differ by less than a quarter of a task; our own hindsight bound on Verified and Terminal-Bench 2.0 is 87 and 50, so the gap there is the insurance the prospective rule buys, not the estimator. Under the looser mean-error criterion every method's bound is far lower, ours included, which is why the two criteria must not be mixed.

	$\varepsilon = 4$ points			$\varepsilon = 3$ points			$\varepsilon = 2$ points		
Bank	weak third	middle	strong third	weak third	middle	strong third	weak third	middle	strong third
Verified	0.95 [0.93, 0.97] (135)	0.90 [0.88, 0.93] (135)	0.97 [0.96, 0.98] (70)	0.93 [0.91, 0.95] (177)	0.90 [0.87, 0.92] (197)	0.97 [0.95, 0.99] (140)	0.82 [0.80, 0.87] (185)	0.67 [0.64, 0.70] (200)	0.92 [0.91, 0.94] (190)
Lite	0.96 [0.93, 0.99] (52)	0.98 [0.95, 0.99] (69)	0.90 [0.86, 0.94] (66)	0.98 [0.96, 1.00] (86)	0.93 [0.89, 0.96] (115)	0.91 [0.86, 0.94] (115)	0.98 [0.97, 1.00] (120)	0.92 [0.89, 0.96] (160)	0.93 [0.90, 0.96] (163)
CORE	0.96 [0.93, 0.99] (22)	0.98 [0.94, 1.00] (31)	0.92 [0.89, 0.96] (34)	0.99 [0.97, 1.00] (27)	0.98 [0.95, 0.99] (34)	0.90 [0.86, 0.95] (37)	0.99 [0.93, 1.00] (34)	0.98 [0.94, 1.00] (39)	0.94 [0.90, 0.99] (40)
τ-bench	0.99 [0.97, 1.00] (39)	0.96 [0.92, 0.99] (42)	0.96 [0.93, 0.99] (42)	0.97 [0.95, 1.00] (42)	0.95 [0.93, 0.99] (44)	0.99 [0.97, 1.00] (45)	0.99 [0.97, 1.00] (44)	0.96 [0.95, 1.00] (46)	0.99 [0.98, 1.00] (46)
TB 2.0	0.91 [0.89, 0.94] (58)	0.94 [0.92, 0.97] (72)	0.86 [0.82, 0.90] (63)	0.78 [0.75, 0.82] (78)	0.89 [0.86, 0.92] (86)	0.83 [0.77, 0.86] (84)	0.52 [0.46, 0.58] (89)	0.74 [0.70, 0.78] (89)	0.61 [0.55, 0.65] (89)

Table D.9: **Realized reliability of the stopping rule by agent ability.** Test agents of all 30 splits are pooled and cut into terciles by their true full-bank pass rate. Each cell: the share within ε in that tercile, a 95% bootstrap interval over splits, and in parentheses the mean stopping count.

D.5 Is the realized reliability uniform across agents?

The reliability the stopping rule states is a share over held-out agents, and a share can hide a systematic miss: a rule that covers strong agents at 1.00 and weak ones at 0.80 averages 0.90. Table D.9 pools the test agents of all 30 splits, cuts them into terciles by their true full-bank pass rate, and reports the share within ε in each tercile, with the mean stopping count. The rule, the splits and the totals are those of Table 2; the table only disaggregates them.

At four points every tercile on every bank is at or above 0.90 except the strongest third of Terminal-Bench 2.0, at 0.86 with interval $[0.82, 0.90]$, and the misses are not concentrated in the weak agents: on SWE-bench Verified the weakest third is covered at 0.95 and the strongest at 0.97, with the middle third lowest at 0.90. Where a tercile falls well below the stated level, the cap or the bank has bound. At two points on Verified the middle third stops at the 200-task cap on average and is covered at 0.67; on Terminal-Bench 2.0 at two points every tercile exhausts the 89 tasks, which is the *all* cell of Table 2. The rule therefore fails where it cannot spend more, not for a class of agents it misjudges, and the strong-agent shortfall on Terminal-Bench 2.0 is the one place the pooled share overstates what a particular kind of agent receives.

D.6 C7: repeats must be modelled, and allocated per agent

Treating the five attempts of a cell as independent draws overstates what a rerun is worth, and the repeat-aware model predicts the remaining attempts better on all 30 splits (Table C.15). The per-task interaction scale is only weakly identified: fitted on two disjoint halves of the training pool, the estimates correlate at 0.50, 0.50 and 0.38 on the three releases, no better than the discriminations and far below the difficulties. It matters anyway, but only for one of the two jobs. Replacing every σ_j by their mean leaves the allocation and the score untouched, at all seven budgets, and makes the prediction of individual reruns significantly worse, by 0.0016 of log-likelihood per run and 0.0009 of Brier score. The existence of the interaction decides whether a rerun is worth buying; its per-task value only sharpens the prediction for a single cell. The gain is the allocation and not the estimator. Holding the allocation uniform and changing only the estimate, from the empirical mean of the attempts run to the repeat-aware posterior, moves the score error by 0.00 to 0.09 percentage points; changing the allocation on top of that moves it by a further 0.26 to 0.34, so between 74% and 100% of the improvement above one full pass comes from deciding which cell to run again. At one full pass neither helps, which is the same census point the budget sweep identifies. Given the model, the allocation matters more than the schedule: deciding per agent which cell to run again saves about a fifth of the runs against uniform reruns (Table C.19), while fixing a repetition count per task in advance, using the same estimated interaction scales, beats equal counts at only one of six budgets. Finally the allocation has to be guarded: when the item parameters come from a different release, deciding reruns is worse than uniform reruns, and the misfit check restores it (Table C.18).

D.7 The same comparison under a matched reference

Scoring an estimate against a record that contains the runs it observed makes the estimate look better than it is, so we repeat the comparison with a reference the method cannot see. Each cell's five Terminal-Bench attempts are split per split into an acquisition pool of two and a reference pool of three. Every method may draw runs only from the acquisition pool, and the target is the pass rate over the reference pool: an independent three-attempt evaluation of the whole bank. The protocol's own arm is then directly comparable, because one complete single-attempt pass costs N runs from the same pool and is scored against the same target (Table D.10).

Release	Runs spent	$0.25N$	$0.50N$	$0.75N$	$1.00N$	$1.50N$	$2.00N$
TB 2.0	one or two complete passes	7.30	5.00	3.59	2.76	2.50	2.13
	our selection, one run	3.82	2.97	2.71	2.70	2.70	2.70
	our selection and reruns	**3.82**	**2.89**	**2.51**	**2.28**	**2.14**	**2.13**
TB 2.1	one or two complete passes	6.92	4.37	3.54	2.64	2.30	2.11
	our selection, one run	3.85	2.96	2.58	2.60	2.60	2.60
	our selection and reruns	**3.85**	**2.94**	2.75	**2.38**	**2.17**	**2.11**
TB 4.0	one or two complete passes	8.11	5.54	3.84	3.41	3.35	2.69
	our selection, one run	4.79	3.73	3.35	3.23	3.23	3.23
	our selection and reruns	4.79	3.92	3.59	3.30	**2.79**	**2.69**

Table D.10: **A matched reference.** Each cell's five attempts are split into an acquisition pool of two and a reference pool of three; every method may draw only from the acquisition pool, and the target is the pass rate over the reference pool. The *one or two complete passes* row is the protocol itself. Error of the predicted bank score in percentage points, 30 splits.

The pool-size boundary survives the matched reference, and the rest of the claim weakens. On the three thin releases our procedure matches one complete pass at the same number of runs rather than beating it: the difference at $1.00N$ is -0.26 and -0.11 percentage points on releases 2.1 and 4.0, neither significant, and the advantage over that single pass appears only past one and a half passes, 0.36 to 0.73 points at $1.50N$ and $2.00N$. On Terminal-Bench 2.0, with 95 training agents, the advantage over one complete pass is already significant at three quarters of a pass, -0.25 points, and reaches -0.48 at one pass and -0.63 at two passes' worth of runs; against two complete passes at that budget the two tie at 2.13 points, since both have then exhausted the acquisition pool (Table C.17). Below half a pass every release is worse than a full census, by 1.1 to 1.4 points, the same boundary the run-budget sweep found.

Selection still earns its place at small budgets, but against the same budget rather than against a census: a quarter of a pass spent on our selection gives 3.85, 4.21 and 4.79 points against 6.92, 6.78 and 8.11 for the same runs spent in the protocol's order. What the matched reference removes is the unqualified claim that a subset can beat rerunning the benchmark. Under a reference that shares no runs with the estimate, that holds on Terminal-Bench 2.0 from three quarters of a pass onward, and on the three thin releases only at one and a half passes or more.

The boundary is the training pool, shown without the cross-release confound. Attributing the difference between releases to the number of training agents is not safe on its own, because the releases also differ in their task banks and in which agents they contain. We therefore thinned the pool inside one release. On Terminal-Bench 2.0, holding the task bank and the same 32 held-out agents fixed, we subsampled the training pool to 9, 13, 17, 30 and 95 agents, 30 draws each, and repeated the comparison against the protocol's own order (Table D.11). At one complete pass the advantage over a census grows monotonically with the pool, from $+0.15$ and $+0.04$ points at nine and thirteen agents, neither significant, to -0.16, -0.43 and -0.58 at 17, 30 and 95, all significant. Nine to thirteen agents are the pool sizes of the thin releases, and at those sizes Terminal-Bench 2.0 reproduces their behaviour with their confounds removed. Below a full pass our selection wins at every pool size, by 1.18 to 2.31 points, because there the comparison is against the same budget spent in the protocol's order rather than against a census.

What the pool is made of matters much less than how large it is. Benchmark pools are not samples of independent systems. One scaffold family supplies 41% of the CORE-bench Hard pool, 28% of τ-bench airline, 25% of Terminal-Bench 2.0 and 23% of SWE-bench Verified. Variants of one scaffold answer alike, so a concentrated pool may count the same evidence several times when the item parameters are fitted. We tested this at matched pool size. One arm keeps a single agent per scaffold family, which leaves 20% to 57% of the training agents. The other arm draws the same number of training agents at random, ignoring families. Everything else is identical. Across five benchmarks and three budgets, 12 of the 15 cells tie. Deduplication is better on CORE-bench Hard at two tasks, 9.02 against 10.52 points, and worse on Terminal-Bench 2.0 at 9 and 18 tasks, 4.63 against 4.30 and 3.24 against 2.91. We registered adoption on deduplication being better in half the cells and worse in none, so we do not adopt it, and we report that it is harmful on the one benchmark where it

Calibration agents	44 runs ($0.5N$) Ours	difference	89 runs ($1.0N$) Ours	difference	178 runs ($2.0N$) Ours	difference
9	3.63	-1.18*	2.37	+0.15	1.46	+0.08
13	3.35	-1.45*	2.25	+0.04	1.37	-0.01
17	3.07	-1.74*	2.05	-0.16*	1.21	-0.18*
30	2.73	-2.08*	1.79	-0.43*	1.11	-0.27*
95	2.50	-2.31*	1.64	-0.58*	0.96	-0.43*

Table D.11: **Thinning the training pool inside one release.** Terminal-Bench 2.0, with the bank and the same 32 held-out agents fixed and the pool subsampled to the size shown, 30 draws per size. *Ours*: error of the predicted bank score in percentage points. *Difference*: ours minus the protocol's own order at the same budget. */†: ours significantly lower/higher.

is not neutral. The useful statement for a maintainer is the negative one: at a given pool size, how many of the agents are variants of one scaffold is not what limits calibration. The pool size is. We do not have a tested explanation for the two benchmarks that move, and we do not offer an untested one.

D.8 What an operator actually pays for, and why we still count runs

The task count is not the bill, and we measure how far apart the two are, but we do not denominate our budget in money. A language model answers every item with one forward pass, so the number of items is the cost. A coding agent works in a container for minutes to an hour, and the tasks of one bank differ by far more than they differ in information. In the 5,940 recorded runs of Terminal-Bench 4.0 the mean cost of a task ranges over a factor of 33.6, with a median of \$7.4 and a ninetieth percentile of \$22.5, and the median run takes 38 minutes. Running every task once costs \$97 on Terminal-Bench 2.1 and \$690 on 4.0.

Selecting tasks by information per dollar does not pay. We therefore repeated the selection experiment with the budget counted in dollars rather than tasks, using each task's mean cost among the training agents as its expected price, and charging each test agent its own recorded cost. Selecting by information per dollar changes the bill substantially on one release and not at all on another. On Terminal-Bench 4.0 it reaches a score error of three percentage points for \$100, where selecting by information alone needs \$318, and at a fixed \$69 budget the error falls from 5.03 to 3.18 points. On Terminal-Bench 2.1 it is worse: the same target costs \$43 instead of \$10, and Kendall τ at a fifth of the full cost falls from 0.78 to 0.70, because the cheap tasks there are also the ones that separate agents. Over twelve budget cells the score error is significantly lower in 6 and never higher, while the ranking is significantly worse in 2. Replacing the ratio by the expected reduction in the variance of the predicted score per dollar does not improve on the simple ratio in any cell and is worse in one.

Information is not free, which limits what cost-awareness can buy. Across the three releases the cost of a task and the information it carries are positively correlated, at $+0.25$ to $+0.34$ (rank correlation $+0.26$ to $+0.41$): the tasks that separate agents tend to be the ones that take longer and cost more. The exception is Terminal-Bench 4.0, where the quarter of tasks carrying the most information costs \$8.41 on average against \$10.64 for the bank, and this is exactly the release where cost-aware selection pays. On Terminal-Bench 2.1 the most informative quarter costs more than the average, \$1.38 against \$1.10, and cost-aware selection does not pay.

Taking the cheapest task among the near-maximally informative ones is the safer form, and it is still not safe enough. This constrained rule cannot trade information away by construction: at every step, restrict to the tasks whose information is within 10% (or 25%) of the best available, and among those take the cheapest. It is better than the ratio rule but still not uniformly safe: over three releases, six dollar budgets and two metrics, the 10% band is significantly better in 8 cells and worse in 6, and the 25% band better in 10 and worse in 7. The per-step bound on information does not bound the information of the whole sequence, because the cheap tasks inside a band are not independent of one another (Table D.12).

We therefore keep the budget denominated in runs, and we do not offer a certificate in dollars. Cost-aware selection is not adopted either, because it trades ranking quality for money on one of the three releases. The measurement is the point: on agentic benchmarks the budget that an operator

			Score error (points)			Kendall τ		
Release (full cost)	Budget	Tasks	info	info/$	band	info	info/$	band
TB 2.1 ($97)	$10	8	2.96	3.31	3.57	0.724	0.664	0.715
	$19	18	3.46	3.42	3.88	0.779	0.695	0.770
	$29	29	3.71	3.31	3.87	0.766	0.713	0.735
	$49	50	3.47	2.96	3.42	0.750	0.782	0.750
TB 4.0 ($690)	$69	10	5.03	3.18	4.56	0.781	0.884	0.851
	$138	17	4.06	2.84	4.08	0.838	0.891	0.845
	$207	24	3.88	2.33	3.66	0.872	0.905	0.892
	$345	35	2.86	1.99	2.63	0.935	0.948	0.930

Table D.12: **Selecting tasks when the budget is money.** Info: our selection, which ignores cost. Info/$: information divided by the expected cost of the task. Band: the cheapest among the tasks whose information is within 10% of the best available. Tasks: how many tasks the information rule buys at that budget.

approves is denominated in dollars, a task can cost 33 times another, and the task count that the literature optimizes is not the quantity being spent.

D.9 Per-task evaluation cost

On SWE-bench Verified we recover the per-task dollar cost of 35 submissions from their released trajectories (`instance_cost`); all 35 use the mini-SWE-agent scaffold, so these figures describe one scaffold family. The median task costs $0.224, the mean $0.366, the 90th percentile $0.859 and the maximum $3.29. Failed runs cost 1.38 times as much as successful ones, and a task's cost correlates 0.521 with its difficulty. On Terminal-Bench 2.0, per-step token counts are available for nine submissions from six scaffolds; within each submission, failed runs use a median of 1.91 times as many tokens as successful ones.

What the stopping rule saves in dollars. The SWE-bench Verified costs allow one end-to-end conversion. For each split we replay the per-agent rule with the split's own fit on every test agent among the 35 submissions with recorded costs, and price the tasks it runs at that agent's own recorded cost. Over the 259 agent–split pairs, the four-point rule stops after 90 tasks on average, 18.0% of the bank. These agents stop earlier than the average test agent, which needs 112. The tasks they run cost $42.70 per agent against $195.10 for the whole bank, 21.9% of the bill, a saving of $152 per agent. The dollar share exceeds the task share because the rule's tasks cost more than the average task. At three and two points the rule runs 32.3% and 38.9% of the bank for 39.9% and 48.9% of the bill. These figures describe one scaffold family priced at its own runs. They are a measured saving for that family, not a certificate in dollars.

D.10 What a rule costs when it may use only the agent's own draws

The full-bank score of one agent is the mean of a finite population of N binary task outcomes, so it can be bounded by the sampling design alone, with no item parameters and no latent ability. Sequential evaluation methods for language models take this route, and it is the one thing an agentic benchmark cannot afford, because it cannot touch the record the previous agents have already left. We measured the difference. Tasks arrive in uniform random order, and two bounds are read after every draw: the betting confidence sequence of Waudby-Smith & Ramdas (2024) for sampling without replacement, which is valid at every step simultaneously and so permits optional stopping by construction; and the fixed-sample interval with a finite-population correction, which is the interval a precision-based sequential rule reads (Arviv et al., 2026; Zhou et al., 2026) and which we charge no correction for repeated looks, so its counts are a lower bound on what such a rule spends. Each agent stops at the first budget whose half-width is within the tolerance, over every test agent and 10 task orders.

Neither is viable at these bank sizes (Table D.13). At a four-point tolerance the fixed-sample rule needs 250 of 500 tasks on SWE-bench Verified and 182 of 300 on SWE-bench Lite, against 112 and 63 for our calibrated rule. On the two small banks it stops only after almost everything has been

Table D.13: **Tasks a rule needs when it may use only the agent's own draws.** Mean over every test agent and 10 task orders. *Sequence*: the betting confidence sequence, valid under optional stopping. *Fixed-sample*: the interval a precision-based rule reads, with a finite-population correction and no correction for repeated looks, so its counts are a lower bound; *met* is the share it leaves within ε.

		Sampling only, no task model			TITRATE
Bank (tasks)	ε	Sequence	Fixed-sample	met	tasks
SWE-bench Verified (500)	4	420	250	0.94	**112**
	3	445	318	0.95	**170**
	2	467	396	0.95	**191**
SWE-bench Lite (300)	4	253	182	0.94	**63**
	3	265	217	0.95	**105**
	2	278	254	0.95	**148**
CORE-bench Hard (45)	4	42	41	0.96	**30**
	3	43	43	0.95	**33**
	2	45	44	0.98	**38**
τ-bench airline (50)	4	47	46	0.99	**41**
	3	48	48	1.00	**44**
	2	49	49	1.00	**46**

run: 41 of 45 tasks on CORE-bench Hard and 46 of 50 on τ-bench airline, where our rule spends 30 and 41. The confidence sequence costs more again, 420 of 500 on Verified, which is what an anytime-valid bound pays for covering every step at once; its realized coverage is 0.95 to 0.99 against the 0.95 it states.

The ordering is what the theory predicts and the size of the gap is the measurement: on a bank of a few hundred expensive tasks, a bound computed from one agent's own draws is close to running the whole bank, because the evidence that the other agents have already revealed about the tasks is unavailable to it. On SWE-bench Lite the 63 training agents are worth roughly 120 tasks of evidence, and anytime validity costs about 70 more on top of that. This is why the guarantee we do state is calibrated over agents rather than over task draws (Appendix C.1.3): it spends the training pool, which is the one resource an agentic leaderboard has in quantity and a design-based bound cannot use.

Terminal-Bench 2.0 is left out because the comparison is not defined there. Its reference score averages five attempts per task, and a rule that draws one label per task is estimating the mean of a different population; the confidence sequence covers the reported pass rate in 0.17 of runs, and the fixed-sample rule leaves 0.80, 0.72 and 0.57 of agents within four, three and two points while believing it has stopped on target.

D.11 TWO DESIGNS THAT DID NOT SURVIVE

Spending part of the budget on random tasks costs more than it buys. A model-based estimate is only as good as the model. The standard remedy is to draw part of the sample at random and correct the model's prediction with the residuals on those tasks, which makes the correction unbiased by design and gives an interval with a coverage guarantee. We gave a quarter of the budget to tasks drawn uniformly from those not selected adaptively, and estimated the full-bank score with the difference estimator and with PPI++ (Angelopoulos et al., 2023a;b). Table D.14 reports the result: the score error is one and a half to three times that of the purely model-based estimate at the same budget, and the 95% intervals are 17 to 90 percentage points wide, because 2 to 25 audit tasks cannot estimate a residual mean. At these budgets, design-based validity is not affordable.

An informative prior from the agent's model and scaffold helps only where the labels are dense. Benchmarks record which language model and which scaffold each agent uses. Following the decomposition of Ge et al. (2026), we fit $\theta_i \approx \mu + \alpha_{\mathrm{model}(i)} + \gamma_{\mathrm{scaffold}(i)}$ on the training agents by ridge regression, and used it as the prior for a test agent whose model and scaffold have both been seen. On Terminal-Bench 2.0, where every agent carries one model label and 127 agents cover 45 models, this is a clear gain: Kendall τ at four to nine tasks rises from 0.818 to 0.851 and the score

		Score error (points)			Difference estimator	
Benchmark	Tasks	model only	difference	PPI++	coverage	width
Verified	50	2.76	6.00	6.54	0.91	29
	100	2.23	3.54	3.77	0.92	17
Lite	30	2.61	7.12	8.28	0.90	35
	60	1.90	4.46	4.84	0.92	21
CORE	4	9.13	18.41	20.99	0.69	72
	9	5.33	12.88	16.19	0.66	48
τ-bench	5	9.19	23.10	27.09	0.64	90
	10	6.88	19.34	23.68	0.64	72
TB 2.0	9	4.53	15.08	22.71	0.70	59
	18	3.31	9.74	12.62	0.77	40

Table D.14: **Spending a quarter of the budget on tasks drawn at random**, so that the correction is unbiased by design and the interval has a coverage guarantee. Model only: the same budget spent adaptively.

error falls from 5.08 to 4.14 points, and for the agents the prior applies to, from 4.95 to 3.53. On SWE-bench Verified it is a clear loss: 131 labelled agents carry 89 distinct models, most of them appearing once, spread over three years of releases, and for the agents the prior applies to the error rises from 5.56 to 7.17 points. Gating the prior on the support of the label and on the cross-validated R^2 of the decomposition removes some of the damage but also most of the gain. Over eight settings the prior is significantly better in 7 cells and significantly worse in 7, so we do not adopt it. The regularity is worth stating: this kind of prior is worth its risk only on a benchmark whose agent labels are dense and consistent.

D.12 What survived on the benchmarks we did not design on

The additional benchmarks test selected decisions from development. Their role differs by decision: the shrinkage rule was revised during development, on a release we do not report, whereas the misfit check was fixed before its confirmation on release 4.0.

The shrinkage rule has additional development evidence. On the two further Terminal-Bench releases, 1.0 and 4.0, over two budgets and four metrics, our predictions of the tasks an agent has not run are significantly better than FLUID's in 13 of 16 cells, tied in 3, and worse in none; the ties are the AUC and accuracy on release 4.0, with 13 training agents, where FLUID's hierarchical fit orders tasks as well as ours.

The misfit check. Deciding reruns was better than uniform reruns on Terminal-Bench 2.0 and worse when the item parameters came from another release. The check was designed on those two settings and confirmed on Terminal-Bench 4.0, which we had not used: it is never significantly worse than uniform reruns and significantly better at 99 and 264 runs.

Temperature scaling and the decomposition prior were rejected by the same procedure, reported above.

Where the method degrades. The pool-size experiment of Table C.6, the rerun boundary between 13 and 17 training agents on Terminal-Bench 2.0, and the failure of ATLAS's pipeline below fifteen agents together locate the regime our procedure is for: a benchmark with roughly twenty training agents or more.

D.13 Which internal validation chooses the shrinkage

The best shrinkage depends on the benchmark and on the budget, and no rule we could validate internally tracks both. Sweeping a fixed weight over the same splits shows two patterns that pull against each other. On SWE-bench Verified, the largest training pool at 130 agents, the lightest weight is best at every budget: 1 gives 2.28, 1.76, 1.49 and 0.95 points where 100 gives 2.74, 2.05, 1.65 and 0.97. On the thinner pools the best weight grows with the budget: 3, 3, 10 and 100 on CORE-bench

Benchmark	Calibration agents	Splits choosing λ				
		1	3	10	30	100
SWE-bench Verified	130	30	0	0	0	0
SWE-bench Lite	63	2	8	8	7	5
CORE-bench Hard	49	10	4	7	6	3
τ-bench airline	35	9	9	2	6	4
Terminal-Bench 2.0	95	27	3	0	0	0
Terminal-Bench 1.0	15	1	3	7	10	9
Terminal-Bench 4.0	13	30	0	0	0	0

Table D.15: **The shrinkage strength chosen by internal validation on the training agents.** The largest pool and the smallest both choose the lightest weight in every split; the three mid-sized pools spread across the grid, and Terminal-Bench 1.0 leans to the strongest. Pool size alone does not decide the choice. Nothing about the test agents enters this choice.

Hard, 30, 100, 100 and 100 on τ-bench airline, and 3, 100, 100 and 10 on Terminal-Bench 2.0. This growth is a pattern in the sweep, not a mechanism we tested, and it does not hold on the largest pool, where the best weight is flat.

Our rule validates at a tenth of the bank and applies the result at every budget, which matches the best weight on Verified and undershoots it on Terminal-Bench at half the bank. We tried four ways to do better and none of them was safe: selecting by score error instead of log loss, selecting at the deployed budget, doing both, and raising the internal folds from three to ten. Each fixed Terminal-Bench's cell, 2.03 to 1.91 points, and each cost more elsewhere, up to 0.37 points on Verified at 150 tasks, and ten folds behaved exactly like three, which rules out selection variance as the cause. We therefore keep the simple rule and report the cell it loses as the measured limit of what internal validation on this many agents can choose.

There is a structural reason why no single diagnostic fixes this. Thinning the training pool inside one benchmark and measuring both the split-half correlation of the discriminations and the best fixed weight at that pool size, the two move together in opposite directions depending on the budget. At a tenth of SWE-bench Verified the correlation between them is -0.54 with interval $[-0.72, -0.30]$: a pool whose discriminations reproduce better wants less shrinkage, because selection is what the budget buys. At half the bank it is $+0.38$ with interval $[+0.15, +0.57]$, and on Terminal-Bench 2.0 $+0.57$: there the estimate is dominated by the model's prediction for the tasks left unrun, a flatter model is steadier, and only a pool large enough to place the difficulties accurately can profit from flattening. The right amount of shrinkage therefore depends on the pool and on the budget with opposite signs, which is why a rule that reads one number off the training agents cannot track it.

What the rule actually picks. Table D.15 counts the weight chosen by the internal validation on each benchmark. On SWE-bench Verified all 30 splits choose the lightest weight, which is the weight the sweep says is best there, so the rule is both decisive and right on the largest pool. On the three smaller pools the choice is spread across the whole grid: 2, 8, 8, 7 and 5 splits on SWE-bench Lite, and 9, 9, 2, 6 and 4 on τ-bench airline. The rule is not merely sometimes wrong on a thin pool; it is unstable there, which is the same weak identification the split-half correlations show.

How much the grid matters around a chosen value can be read from the fixed-weight sweep of Table D.2 in Appendix D.1, at a fifth of the bank. Moving λ one grid step from 10, to 3 or to 30, changes τ by at most 0.005 on four of the five banks. On τ-bench airline it changes τ by 0.024 and 0.020, and there τ rises steadily with λ, from 0.550 at 0 to 0.661 at 100. The row without a cap and without a penalty lies below every capped row on all five banks.

The shrinkage strength is chosen by internal validation on the training agents, and there is a choice of what that validation measures. Two quantities are natural: the log loss of the predictions on tasks an agent has not run, and the error of the certificate itself. They do not pick the same value, and the difference matters. Selecting by log loss gives the better probabilities and the better score, and a certificate whose realized reliability collapses on the smallest bank, to 0.43 on τ-bench airline at a stated 0.95. Selecting by the certificate's own validation restores it, to 0.77 there and 0.883 on average, and costs 0.33 points of score error on SWE-bench Verified and both calibration comparisons against FLUID.

We therefore use each for what it is for: the test administration and the certificate take the strength chosen by the certificate's validation, and the probability and score outputs are computed from parameters refitted with the strength chosen by log loss, on the same recorded responses. Both choices use training agents only. Against FLUID this leaves one loss in 64 cells, the AUC on CORE-bench Hard at nine tasks, where the single-rule alternatives leave two and five. The cost of the split rule is visible on Terminal-Bench 2.0, where the score error rises from 4.29 to 4.43 points at nine tasks and from 2.94 to 3.01 at eighteen, both still tied with ATLAS's estimator. Appendix B.6 reports the the fixed weight of 10, selection by log loss and selection by the certificate's own validation side by side; the split rule of this section combines the last two.

E ADDITIONAL RELATED WORK

Stochastic agent outcomes. τ-bench reports pass^k, the probability that all of k attempts succeed, as a reliability measure (Yao et al., 2024). Mustahsan et al. (2025) quantify run-to-run inconsistency of agents with the intraclass correlation and recommend fixed numbers of trials. Miller (2024) recommend resampling each question to reduce measurement error. AgentAssay adapts the number of trials to behavioural variance for regression tests of a single agent workflow (Bhardwaj, 2026). These works measure run noise or choose one number of trials for all tasks. We model run noise jointly with ability and task difficulty, and use the model to decide which cell to rerun.

Psychometrics. Classical test theory defines a true score as the mean over hypothetical independent retakes (Lord & Novick, 1968). Generalizability theory decomposes score variance into persons, items, occasions and their interactions, and its decision studies choose how many items and occasions to use (Cronbach et al., 1972; Brennan, 2001). Equation 19 is a decision study of this kind. Many-facet Rasch models add facets to the Rasch model (Linacre, 1989). Our response model adds a normal person–item effect to the 2PL logit, which is the form testlet response theory uses for the dependence among items in a shared passage (Bradlow et al., 1999), and a special case of random item models (De Boeck, 2008). What differs is what the effect represents and what it is used for: for agents it is the match between one agent and one task, it is identified because repeated attempts are real rather than hypothetical, and we use it to decide which cell to run again. For people, the person–item interaction and the occasion effect are confounded in one response per item. For agents, independent retakes are real, so both can be estimated and a retake becomes an action that an adaptive test can choose.

Ranking uncertainty. Resolution diagnostics and rank intervals quantify benchmark uncertainty (Kotawala, 2026; Neuhof et al., 2026a;b). Continuous monitoring can stop an evaluation once a comparison is resolved (Balkır et al., 2026). Our certificate instead fixes the budget before a test agent runs, for a requested whole-cohort correlation.

Statistical guarantees for evaluation. A parallel line gives evaluation formal guarantees rather than measured reliability. Wang et al. (2025a) and Wu et al. (2026) give frequentist coverage for a model's accuracy at a fixed query budget, the latter borrowing strength from previously evaluated models through a Bayesian factor model and selecting questions actively. Our work differs from Wu et al. (2026) in three respects: it calibrates on a pool of tens of agents with shrinkage of the task parameters, it derives a per-agent budget from a score tolerance, and it models repeated attempts; we run no empirical comparison with their method. Zhou et al. (2026) and Hsu & Shekhar (2026) go sequential: both build anytime-valid intervals, which stay valid under optional stopping, and stop when the interval reaches a target width. Arviv et al. (2026) stop on diminishing returns and a minimum detectable effect size, for ranking and model selection rather than for a stated score tolerance, and Pilditch (2026) stop when a Bayesian posterior is precise enough, without a frequentist guarantee and without a calibrated item bank. Kargi & Salinas (2026) calibrate Elo ratings by split conformal prediction over held-out models, which is the exchangeability unit our own guaranteed variant uses, though the estimand there is a ranking rather than a score.

Two differences decide why we do not adopt this route wholesale. The first is the size and price of the bank. Zhou et al. (2026) evaluate on SST-2, AG News and MMLU, pools of 68,000 to 128,000 single-response items, and saving 54–62% of them still leaves thousands of responses; our banks hold 45 to 500 tasks, each a container run of minutes to an hour. Appendix D.10 measures what a design-based bound costs at that size: 253 of 300 tasks at a four-point tolerance, against 63 for the

calibrated rule. The second is what the bound may use. An anytime-valid bound on one agent is computed from that agent's own draws, so the evidence the other agents have already revealed about the tasks is unavailable to it, and on an agentic leaderboard that evidence is the resource that exists in quantity. Our guaranteed variant therefore calibrates over agents instead (Appendix C.1.3), following risk control applied to whole sequential procedures (Ringel et al., 2024; Jazbec et al., 2024; Wu & Rus, 2026) rather than to a fixed budget.

E-processes and risk-controlled early exiting, contrasted. The two sequential families we cite answer different questions from ours. An e-process tests a hypothesis fixed in advance, for example that a model's accuracy exceeds a threshold, and inverting a family of them gives an interval (Waudby-Smith & Ramdas, 2024). The guarantee holds at every stopping time at once. Zhou et al. (2026) and Hsu & Shekhar (2026) stop such an interval once it reaches a target width. Its evidence is the evaluated model's own draws, so it cannot use what earlier agents revealed about the tasks. Risk-controlled early exiting calibrates one threshold on a population of inputs so that the risk of exiting early stays below a fixed level, marginally over inputs (Ringel et al., 2024; Jazbec et al., 2024; Wu & Rus, 2026). Its loss is a per-input prediction error, not the error of a bank score. Neither family turns a score tolerance into a run count that draws on the training agents. The interval family does imply a count, the first draw at which the half-width is within ε, and Appendix D.10 measures it with the betting confidence sequence: on our banks it is close to the whole bank. Our guaranteed variant borrows the construction of risk-controlled early exiting, with agents in place of inputs (Appendix C.1.3). We do not run the published e-process or early-exit methods themselves, so beyond that design-based measurement the contrast is conceptual.

F Additional Limitations and Development Decisions

This appendix expands Section 7. It records the assumptions the certificate rests on and the alternatives we measured and did not adopt.

F.1 The calibration cohort has to represent the new one

The certificate assumes that the training agents represent the new cohort. Fixing the budget in advance never fails, because the number of tasks is computed from training agents only. What degrades is the reliability attached to that number.

In distribution the realized level averages 0.885 at a stated 0.95 and 0.929 at a stated 0.98. Under our time splits the test agents are a later generation, and the realized level drops to between 0.17 and 0.67 at a stated 0.95 (Section C.1.7).

Ordering the internal splits by date as well is the only mitigation we tried that helps, and it recovers the level only partially. It raises the mean realized reliability over the four targets from 0.58 to 0.83 on Terminal-Bench 2.0 and from 0.42 to 0.67 on SWE-bench Lite. It costs 1.2 to 2.9 times the tasks.

Two natural explanations are ruled out by measurement. The new cohort is not more tightly bunched. Its ability variance is 1.24, 1.78 and 0.62 against 1.00 for the training pool on SWE-bench Verified, Terminal-Bench 2.0 and SWE-bench Lite, and pricing the certificate for the tightest cohort we have seen changes nothing. The information curve is not overstated either. The ratio of the information a later cohort actually accumulates to the information predicted from an earlier one is 0.94 to 1.05, and discounting it changes no cell.

A ranking promise is in the end relative to the cohort an agent is compared against. We therefore also report the certificate in conditional form: for a cohort whose ability spread is at least s, K tasks reach the target. A new model family can change the item parameters, the ability variance and the margin, so a changing benchmark needs recalibration. Score prediction and the task budgets needed to reach a target are much less affected.

F.2 What the certificate does not cover

The certificate covers the aggregate Kendall correlation. It does not cover every pair, and it gives no rank interval for an individual agent. Close neighbours can remain unresolved (Neuhof et al., 2026a;b).

F.3 How precisely the reliability itself is estimated

Reliability is estimated from 30 internal splits, so high quantiles are coarse. A 30-of-30 result has a 95% Wilson lower bound of 0.884. That is a bound, not a guarantee, and Table C.14 shows under-coverage in some cells.

F.4 Repeated attempts exist on one benchmark family only

Only Terminal-Bench publishes repeated attempts. Its later releases have 12 to 22 agents, which is too few to calibrate item parameters on their own. We therefore tested whether the rerun allocation transfers across releases.

We calibrated items on the 127 agents of Terminal-Bench 2.0 and evaluated the 22 agents of Terminal-Bench 2.1 on the same 89 tasks. Our allocation is better than uniform reruns at 89 and 134 runs, at 1.83 against 1.94 and 1.47 against 1.57 percentage points of score error. It is significantly worse at every larger budget, at 1.30 against 1.15 at 214 runs, over 30 draws of the attempt order. Terminal-Bench 2.1 revised tasks and removed attempts its maintainers judged to be reward hacking, so the item parameters of 2.0 do not describe it.

Calibrating instead on 16 agents of Terminal-Bench 2.1 itself, our allocation is better at 356 runs, tied at 134, and worse at 214 runs, at 1.14 against 0.99 for uniform reruns.

The misfit check of Section 3.5 removes the cross-release failure, at 1.01 against 1.30 at 214 runs (Table C.18). It falls back to spending half of each batch uniformly. It does not recover the gain that same-release item parameters would give. Single-attempt benchmarks cannot identify the interaction scale σ_j, so on them our procedure reduces to the 2PL model.

F.5 The diagnostic that triggers a conservative fallback

The misfit check is the one diagnostic the procedure runs on a test agent. It measures how probable the agent's first pass over the bank is under the fitted model: the mean per-task log marginal likelihood ℓ_i of Appendix A.1, with the ability integrated over its prior. Its threshold is the fifth percentile of the same statistic on training agents held out from a single internal fit, so it is set before any test agent runs. When ℓ_i falls below the threshold, the allocation stops trusting the item parameters alone: half of every later batch of reruns follows the uniform order and half follows the expected variance reduction, and the score is still estimated by the repeat-aware posterior. The check guards the allocation of reruns only. The per-agent stopping rule and the cohort certificate have no such check, and we make no claim about how either behaves when the check would fire.

F.6 Our two prediction tables use two different predictive quantities

The per-task table integrates the success probability over the ability posterior. The score table substitutes the posterior mean into the model. We report the measurement rather than pick one.

The posterior average is the textbook estimate, so we tested it on the score table at every budget of all five benchmarks. It is significantly better on Terminal-Bench 2.0 and on τ-bench airline at all five budgets. It ties everywhere on SWE-bench Verified. It is significantly worse on three budgets of CORE-bench Hard and one of SWE-bench Lite. We registered before running it that we would adopt it only if no cell got worse. We therefore keep the substituted estimate in the score table and report this inconsistency instead of choosing per benchmark.

We cannot explain which benchmarks it helps. Integration pulls every prediction toward one half, which suggested that it should help where the bank's pass rate is near one half and hurt where the pass rate is extreme. We registered that prediction and tested it on three benchmarks we had not used, and it failed. Integration is significantly worse at all three budgets of SWE-bench Full, whose mean pass rate is 0.193, as the prediction says, but SWE-bench Full is the only one of the three held-out benchmarks that matched it. We therefore report the measurement and withdraw the explanation.

F.7 Model form and bank size

The response model is unidimensional and binary, although tasks involve several skills and partial progress; a second ability dimension, fitted with its own validated penalty, helped only on the 130-agent pool and was worse than one dimension under the chronological splits, so we keep one. Small banks impose a hard limit: CORE-bench Hard and τ-bench airline need much of the bank at high targets.

F.8 How the baselines were run

Under a time split our ranking advantage is not uniform (Section C.1.7). The released code of FLUID and ATLAS cannot take repeated attempts, so the fairest repeated-attempt comparison is against the random subset, whose code accepts a pass rate. The released tinyBenchmarks pipeline is run at 300 training epochs rather than its default 2000, because the default diverges on our agent pools.