SQUALL 9 NOVEMBER 2026

Evaluating LLMs on a fixed budget when costs are unknown

*How to spend a **fixed** dollar budget to **evaluate new** models **across** benchmarks when each run's **cost is known only after it finishes**?*

Athiya Deviyani (adeviyan@cs.cmu.edu)

How is a new model evaluated **today**?

1
A new model or version is ready

2
Run the evaluation harness

3
…on *every* item on *every* benchmark

4
Report one row of scores

How is a new model evaluated **today**?

1
A new model or version is ready

2
Run the evaluation harness

3
…on *every* **item** on *every* benchmark

4
Report one row of **scores**

What is a score?

A benchmark is a list of items. Model *k*'s score on benchmark *j* is its mean success rate θ_{kj} over those items.

What does one item cost?

For reasoning models and agents, one item is a long run consisting of model calls, tool calls and retries, usually billed by the no. of tokens.

How is a new model evaluated **today**?

1
A new model or version is ready

→

2
Run the evaluation harness

→

3
…on *every* item on *every* benchmark

→

4
Report one row of scores

What is a score?

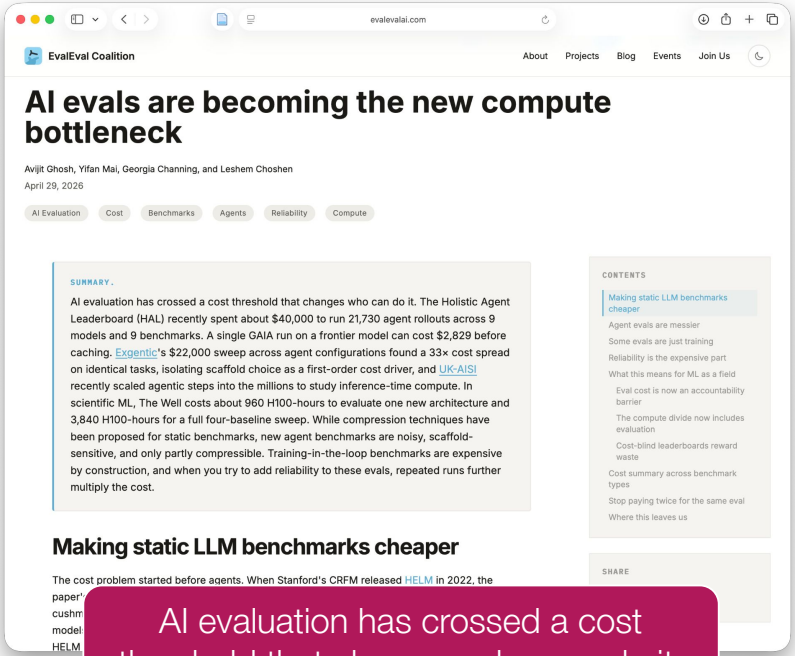
A benchmark is a list of items. Model *k*'s score on benchmark *j* is its mean success rate θ_{kj} over those items.

What does one item cost?

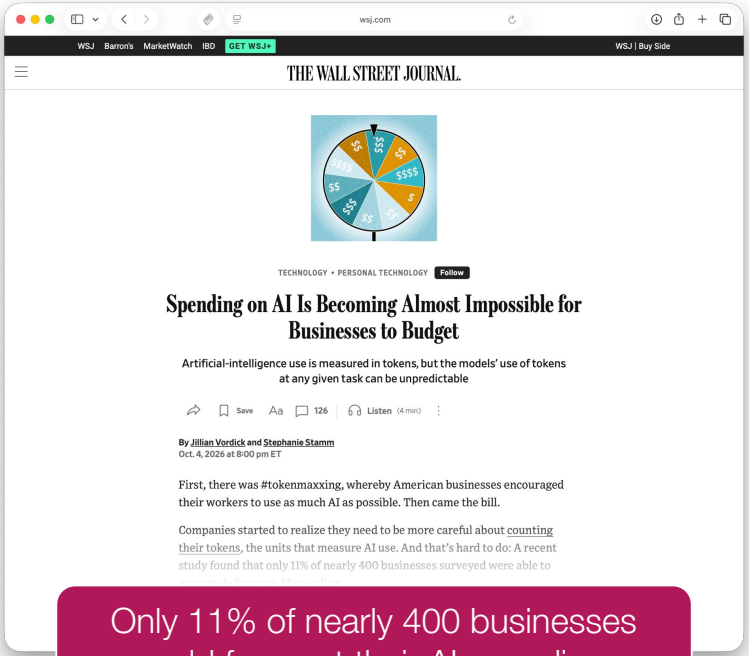
For reasoning models and agents, one item is a long run consisting of model calls, tool calls and retries, usually billed by the no. of tokens.

OR view · Each model or benchmark release brings a batch of jobs. Each item is one job, and its size is its cost.

Everyone is starting to notice the bill

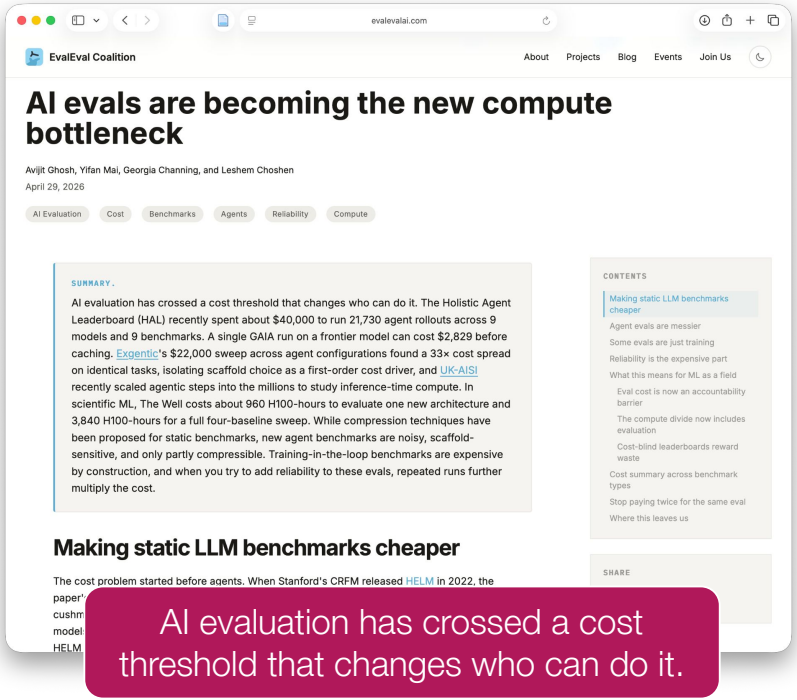


AI evaluation has crossed a cost threshold that changes who can do it.



Only 11% of nearly 400 businesses could forecast their AI spending.

Everyone is starting to notice the bill



It costs a total of **~$40K** for 21,730 agent rollouts on the HAL leaderboard (9 models x 9 benchmarks).

A cost of a single run varies between **$7.80 - $2,929** across all tasks in the HAL dataset.

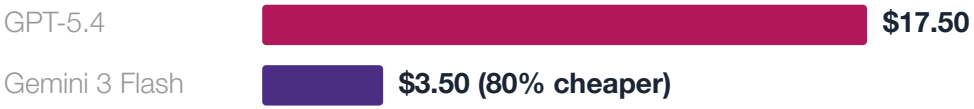
Whoever can pay for the evaluation gets to write the leaderboard.

OR view · Job sizes here span four orders of magnitude, so the average job size tells you little.

Sources: Ghosh et al., EvalEval Coalition blog, April 2026; Kapoor et al., HAL, ICLR 2026

A run's cost is **unknown** until it finishes

List price (input + output, $ per million tokens)

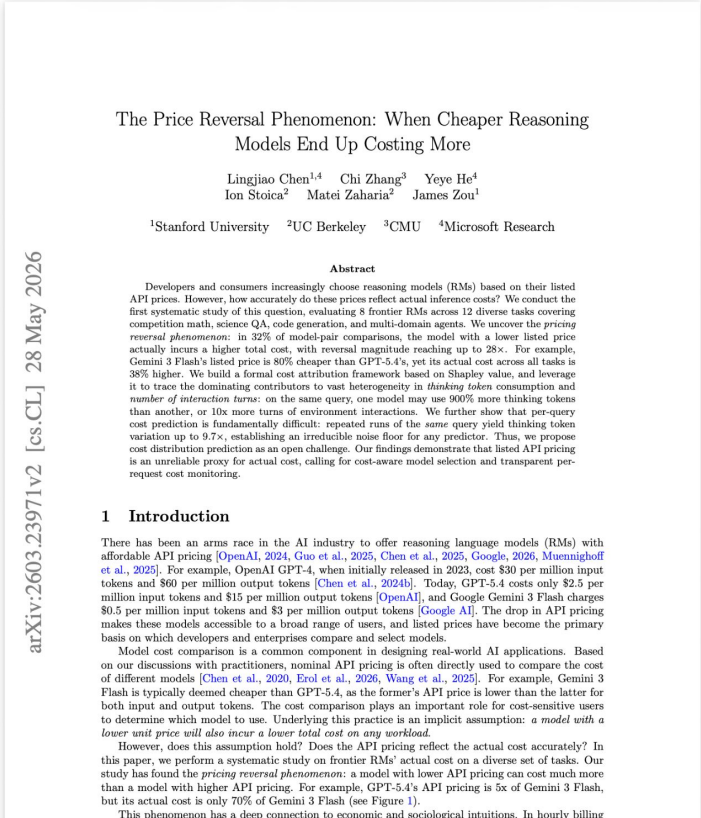


Actual cost, summed over 12 tasks



OR view · Job sizes are unknown until service completes, the setting of size-blind scheduling.

Source: Chen, Zhang, He, Stoica, Zaharia and Zou, "The Price Reversal Phenomenon." arXiv:2603.23971, 2026



arXiv:2603.23971v2 [cs.CL] 28 May 2026

The Price Reversal Phenomenon: When Cheaper Reasoning Models End Up Costing More

Lingjiao Chen[1,4] Chi Zhang[3] Yeye He[4]
Ion Stoica[2] Matei Zaharia[2] James Zou[1]

[1]Stanford University [2]UC Berkeley [3]CMU [4]Microsoft Research

Abstract

Developers and consumers increasingly choose reasoning models (RMs) based on their listed API prices. However, how accurately do these prices reflect actual inference costs? We conduct the first systematic study of this question, evaluating 8 frontier RMs across 12 diverse tasks covering competition math, science QA, code generation, and multi-domain agents. We uncover the *pricing reversal phenomenon*: in 32% of model-pair comparisons, the model with a lower listed price actually incurs a higher total cost, with reversal magnitude reaching up to 28×. For example, Gemini 3 Flash's listed price is 80% cheaper than GPT-5.4's, yet its actual cost across all tasks is 38% higher. We build a formal cost attribution framework based on Shapley value, and leverage it to trace the dominating contributors to vast heterogeneity in *thinking token* consumption and *number of interaction turns*: on the same query, one model may use 900% more thinking tokens than another, or 10x more turns of environment interactions. We further show that per-query cost prediction is fundamentally difficult: repeated runs of the *same* query yield thinking token variation up to 9.7×, establishing an irreducible noise floor for any predictor. Thus, we propose cost distribution prediction as an open challenge. Our findings demonstrate that listed API pricing is an unreliable proxy for actual cost, calling for cost-aware model selection and transparent per-request cost monitoring.

1 Introduction

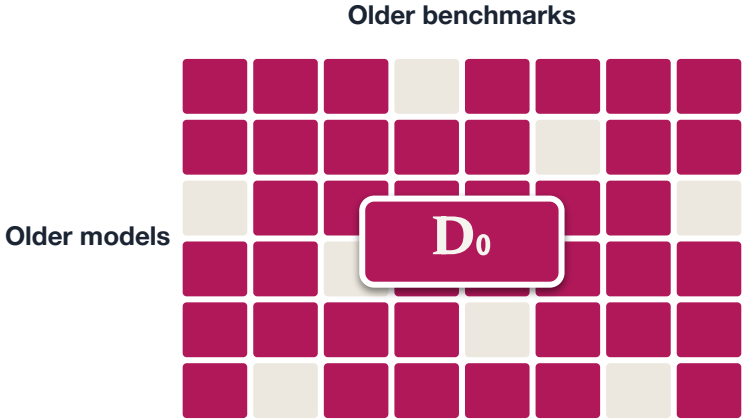
There has been an arms race in the AI industry to offer reasoning language models (RMs) with affordable API pricing [OpenAI, 2024, Guo et al., 2025, Chen et al., 2025, Google, 2026, Muennighoff et al., 2025]. For example, OpenAI GPT-4, when initially released in 2023, cost $30 per million input tokens and $60 per million output tokens [Chen et al., 2024b]. Today, GPT-5.4 costs only $2.5 per million input tokens and $15 per million output tokens [OpenAI], and Google Gemini 3 Flash charges $0.5 per million input tokens and $3 per million output tokens [Google AI]. The drop in API pricing makes these models accessible to a broad range of users, and listed prices have become the primary basis on which developers and enterprises compare and select models.

Model cost comparison is a common component in designing real-world AI applications. Based on our discussions with practitioners, nominal API pricing is often directly used to compare the cost of different models [Chen et al., 2020, Erol et al., 2026, Wang et al., 2025]. For example, Gemini 3 Flash is typically deemed cheaper than GPT-5.4, as the former's API price is lower than the latter for both input and output tokens. The cost comparison plays an important role for cost-sensitive users to determine which model to use. Underlying this practice is an implicit assumption: *a model with a lower unit price will also incur a lower total cost on any workload.*

However, does this assumption hold? Does the API pricing reflect the actual cost accurately? In this paper, we perform a systematic study on frontier RMs' actual cost on a diverse set of tasks. Our study has found the *pricing reversal phenomenon*: a model with lower API pricing can cost much more than a model with higher API pricing. For example, GPT-5.4's API pricing is 5x of Gemini 3 Flash, but its actual cost is only 70% of Gemini 3 Flash (see Figure 1).

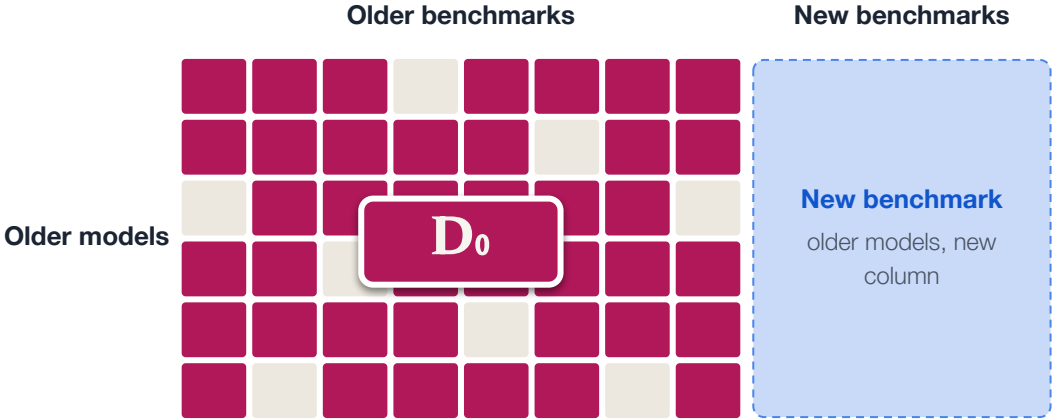
This phenomenon has a deep connection to economic and sociological intuitions. In hourly billing

Problem setting



$\mathbf{D_0}$ is the historical matrix that holds evaluation results for older models and we assume that they are readily available. The dark cells denote evaluations that were run, while the light cells denote evaluations that were never run.

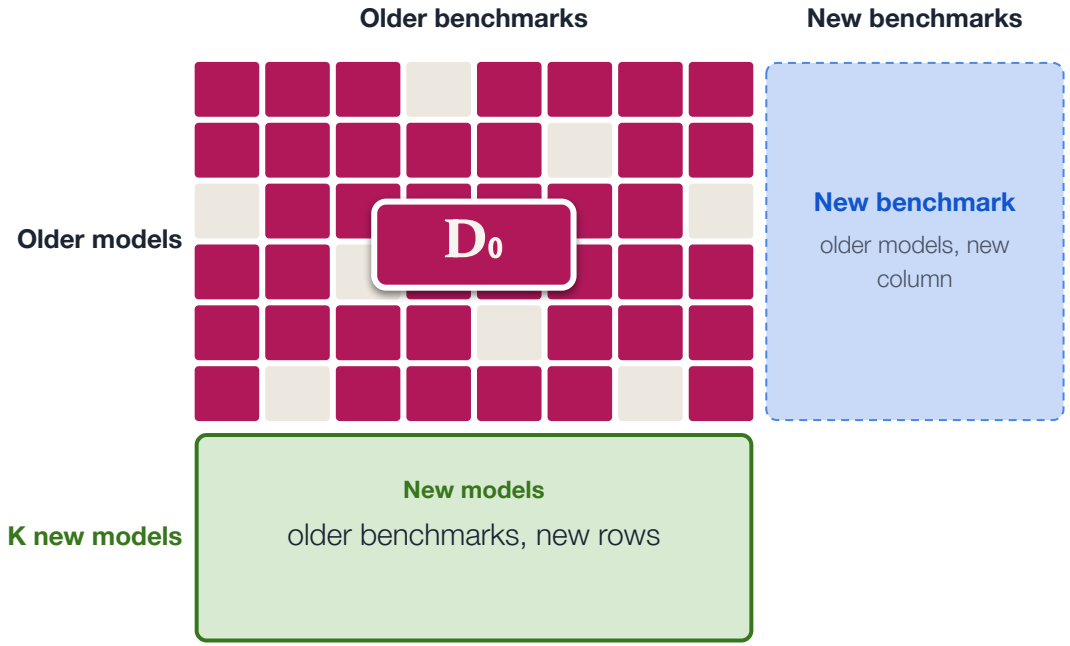
Problem setting



D_0 is the historical matrix that holds evaluation results for older models and we assume that they are readily available. The dark cells denote evaluations that were run, while the light cells denote evaluations that were never run.

There are multiple problem settings covering what components are new

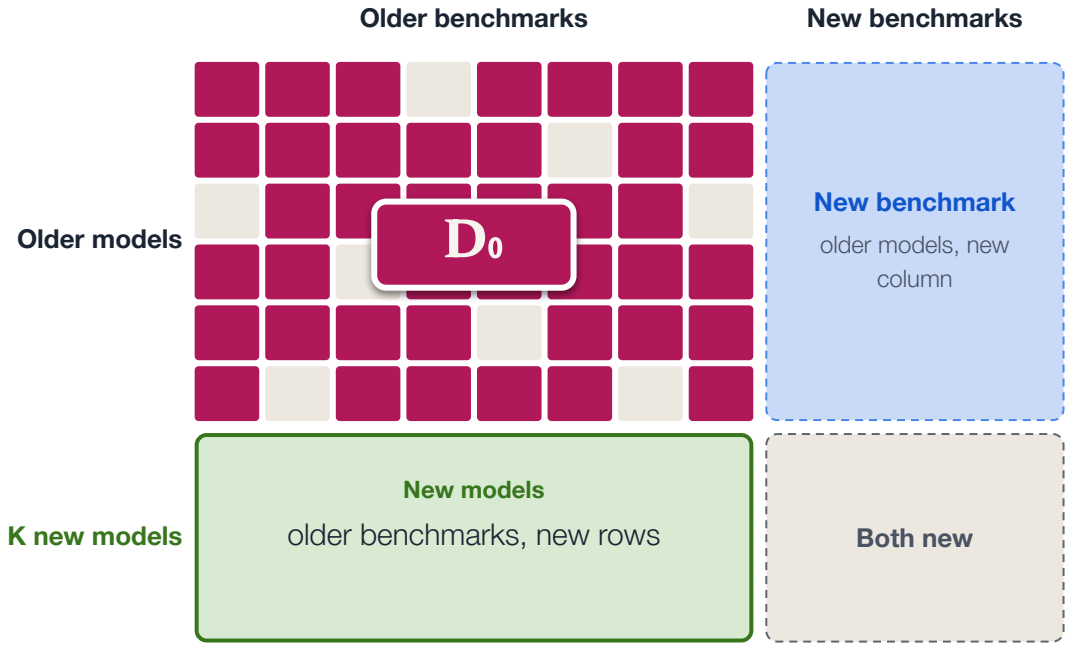
Problem setting



D_0 is the historical matrix that holds evaluation results for older models and we assume that they are readily available. The dark cells denote evaluations that were run, while the light cells denote evaluations that were never run.

There are multiple problem settings covering what components are new

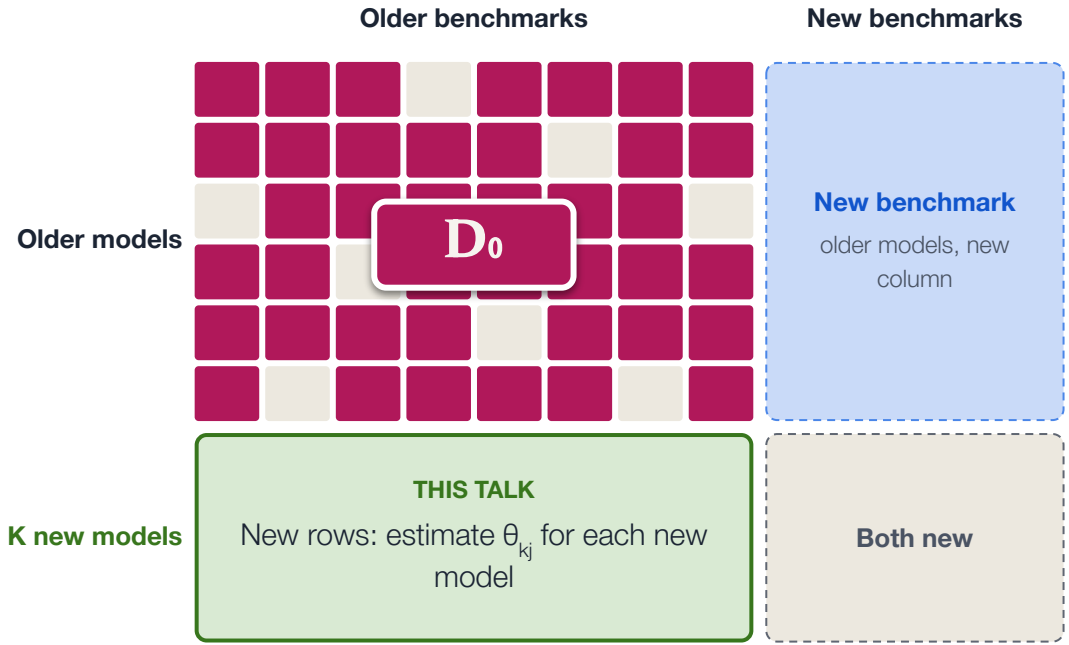
Problem setting



D_0 is the historical matrix that holds evaluation results for older models and we assume that they are readily available. The dark cells denote evaluations that were run, while the light cells denote evaluations that were never run.

There are multiple problem settings covering what components are new

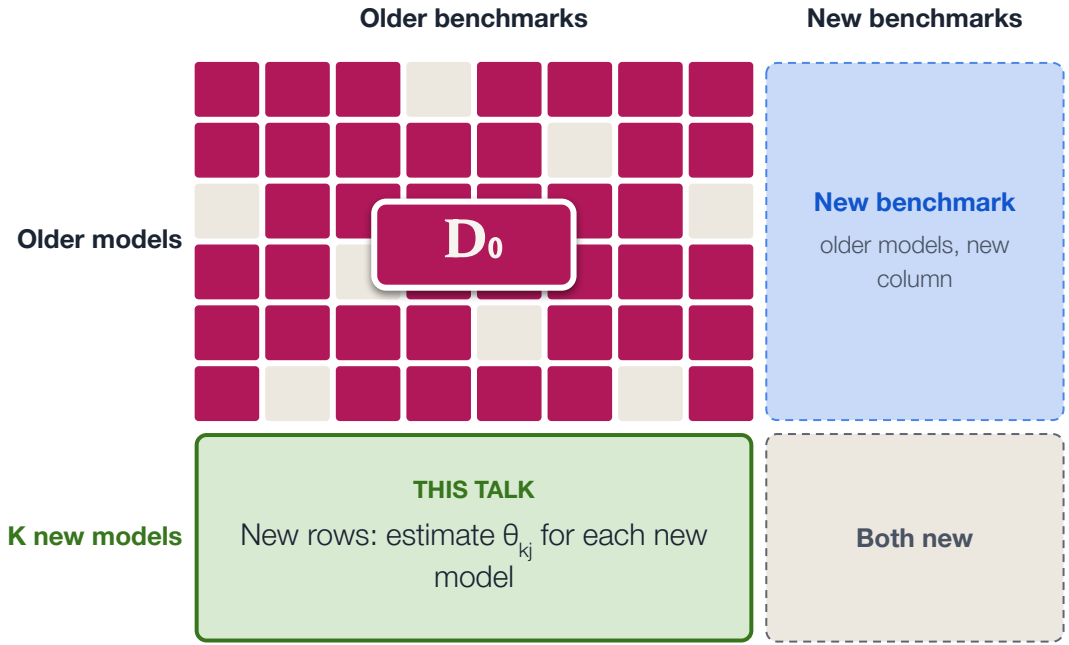
Problem setting



D_0 is the historical matrix that holds evaluation results for older models and we assume that they are readily available. The dark cells denote evaluations that were run, while the light cells denote evaluations that were never run.

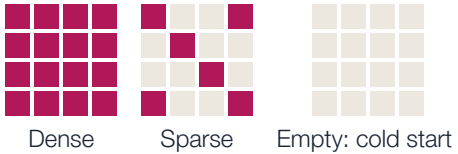
There are multiple problem settings covering what components are new

Problem setting

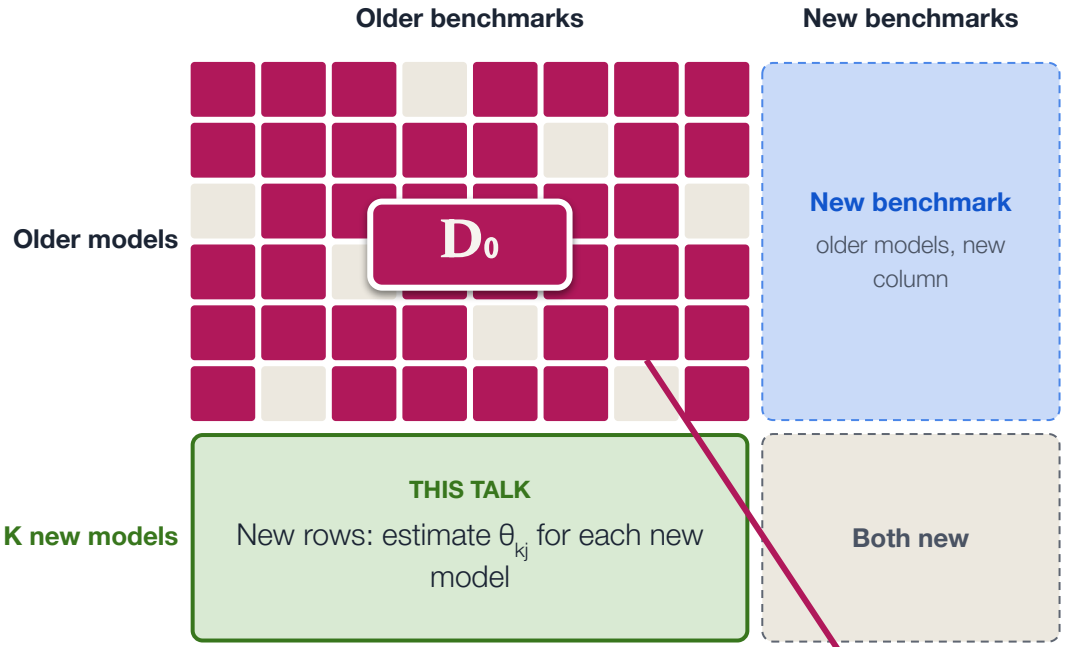


D_0 is the historical matrix that holds evaluation results for older models and we assume that they are readily available. The dark cells denote evaluations that were run, while the light cells denote evaluations that were never run.

There are multiple problem settings covering what components are new, or how much historical data is available.



Problem setting



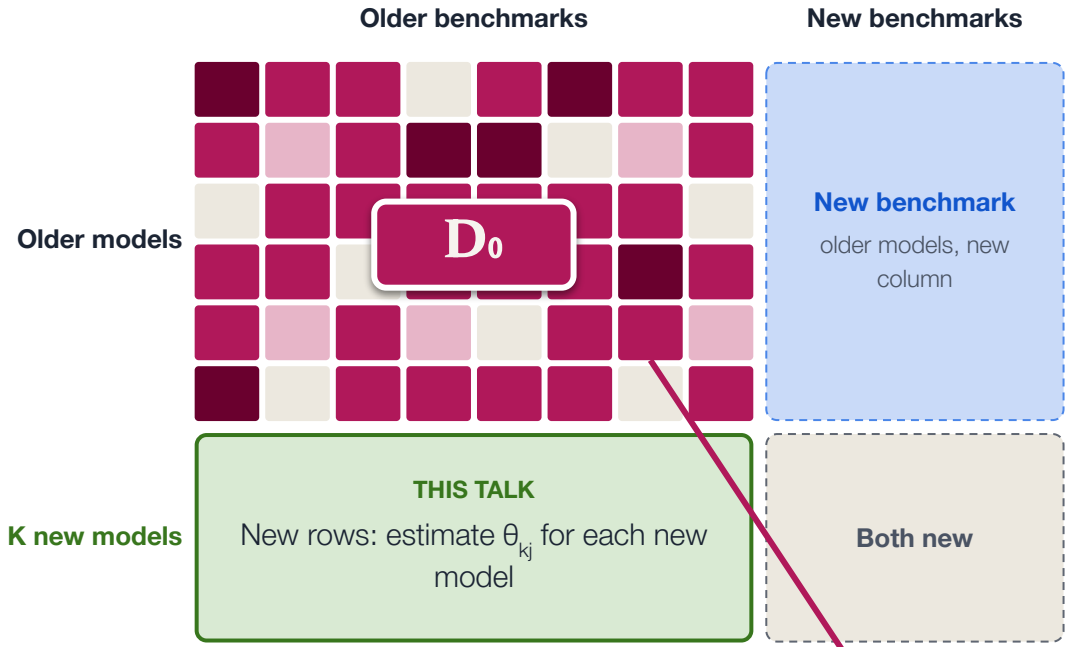
D_0 is the historical matrix that holds evaluation results for older models and we assume that they are readily available. The dark cells denote evaluations that were run, while the light cells denote evaluations that were never run.

There are multiple problem settings covering what components are new, or how much historical data is available.



A benchmark contains a list of items. E.g., a coding benchmark would have tasks such as “code 2sum solution in Python” and the gold item answer.

Problem setting



D_0 is the historical matrix that holds evaluation results for older models and we assume that they are readily available. The dark cells denote evaluations that were run, while the light cells denote evaluations that were never run.

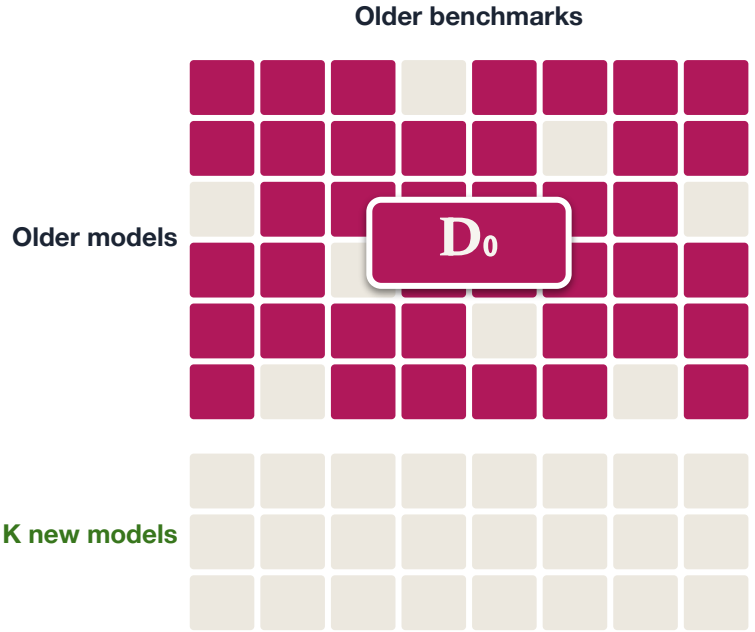
There are multiple problem settings covering what components are new, or how much historical data is available.



A benchmark contains a list of items. E.g., a coding benchmark would have tasks such as “code 2sum solution in Python” and the gold item answer. **A benchmark may be partially evaluated.**

Problem formulation

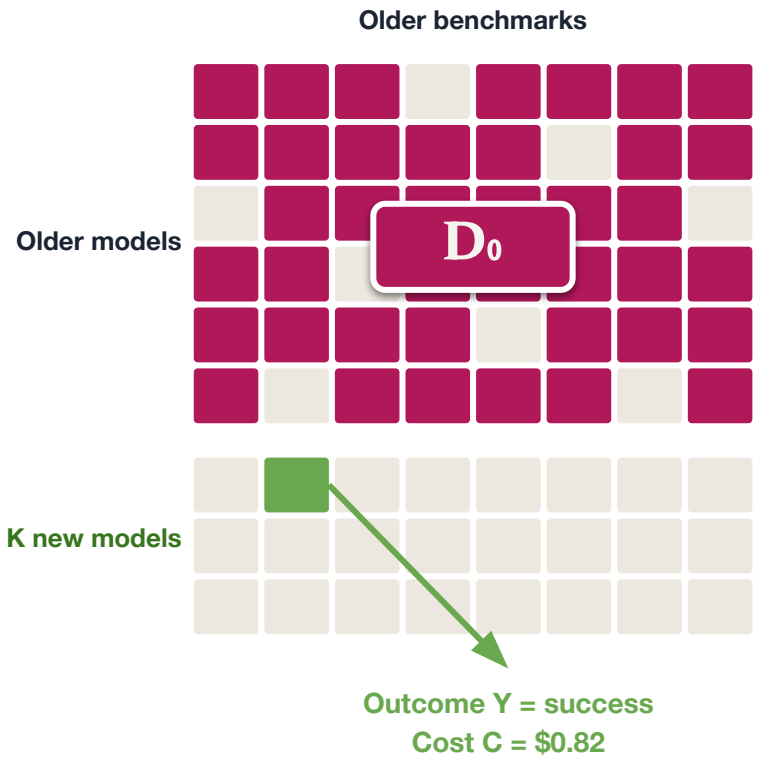
Given. K new models, J benchmarks, a budget of B dollars, and D_0, results for older models (possibly sparse, or empty).



Problem formulation

Given. K new models, J benchmarks, a budget of B dollars, and D_0, results for older models (possibly sparse, or empty).

Each step. Pick a run a = (model k, benchmark j, item i). It returns a score Y and a cost C which are drawn jointly. C is known only when the run finishes.



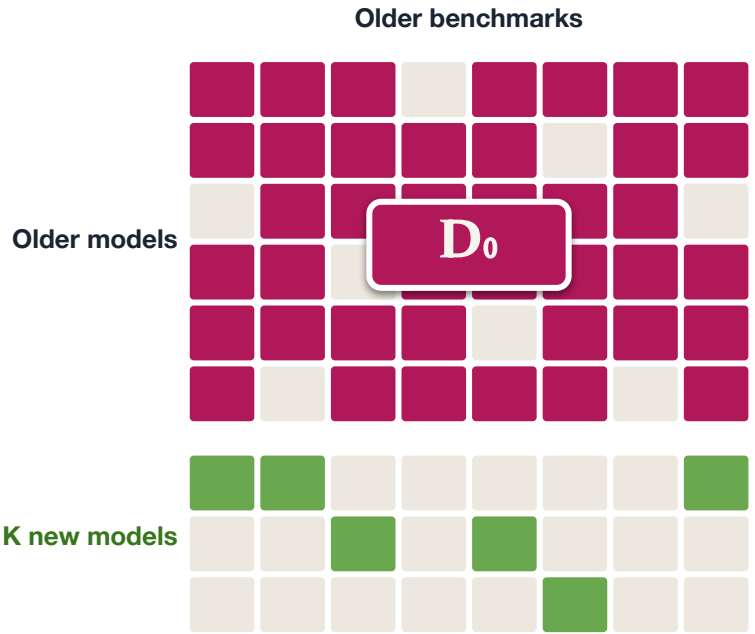
Problem formulation

Given. K new models, J benchmarks, a budget of B dollars, and D_0, results for older models (possibly sparse, or empty).

Each step. Pick a run a = (model k, benchmark j, item i). It returns a score Y and a cost C which are drawn jointly. C is known only when the run finishes.

Runs are atomic. Once started, a run finishes, so the last run can overshoot the budget.

Goal. minimize the expected error of the final estimates, $E[\Sigma (\hat{\theta}_{kj} - \theta_{kj})^2]$



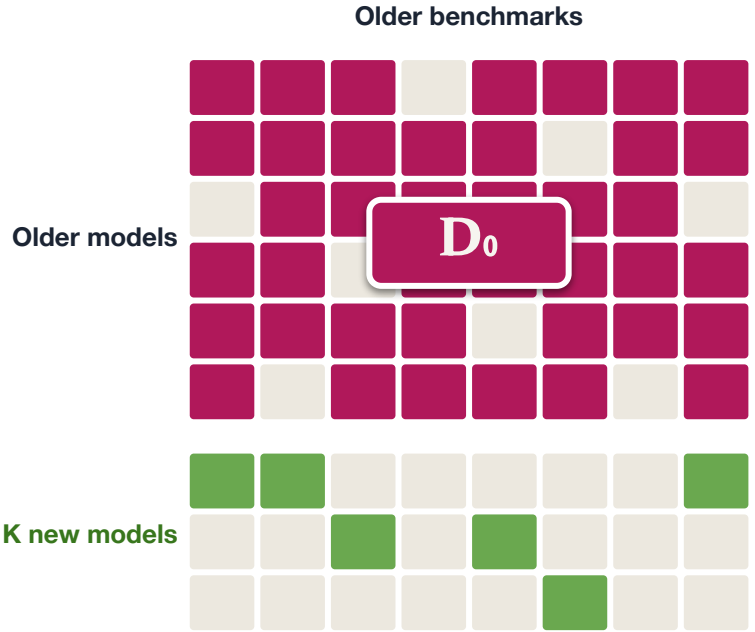
Problem formulation

Given. K new models, J benchmarks, a budget of B dollars, and D_0, results for older models (possibly sparse, or empty).

Each step. Pick a run a = (model k, benchmark j, item i). It returns a score Y and a cost C which are drawn jointly. C is known only when the run finishes.

Runs are atomic. Once started, a run finishes, so the last run can overshoot the budget.

Goal. minimize the expected error of the final estimates, $E[\Sigma (\hat{\theta}_{kj} - \theta_{kj})^2]$



OR view · A batch of jobs, all present at time 0, one server, non-preemptive service, unknown job sizes, and a limit on total work. The objective is estimation error, not response time.

The problem in queuing and OR terms

In LLM evaluation	In queueing and OR
One evaluation run	One job
Run cost	Job size
Cost known only when the run finishes	Unknown job size
A run cannot be stopped partway	Non-preemptive service
Capping a run and keeping a partial result	Preemption
Costs span orders of magnitude across models	Highly variable job sizes
Posterior for one (model, benchmark) pair	Bayesian bandit arm
Dollar budget	Knapsack capacity

Why existing approaches fall short

Existing methods subsample items, fill in the low-rank matrix from older models, or pick runs adaptively. All these miss something:

1 **They budget by *runs*, not *dollars***

One run can cost a cent or a hundred dollars. The one cost-aware method divides information by a fixed historical mean cost, never updated for the new model (Krumdick et al. 2026).

2 **They over rely on historical trends**

New models are often unlike old ones. Score predictions from history miss at the frontier (Zhang, Dorner and Hardt 2025), and so do cost predictions.

3 **They ignore budget risk**

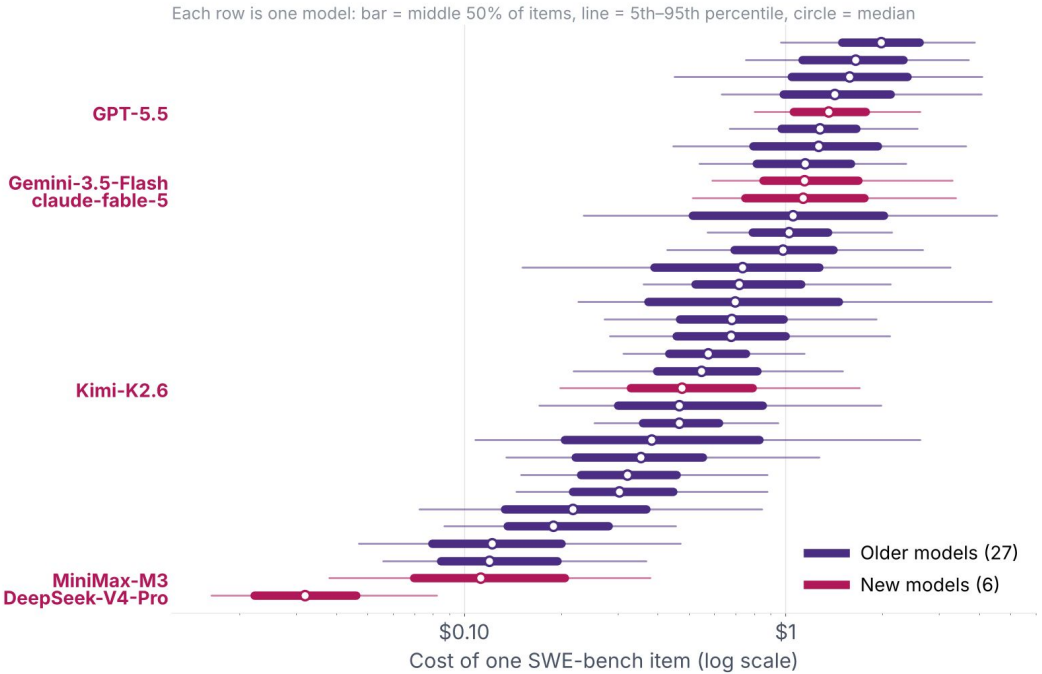
A run's cost is known only when it finishes. One run can use up most of what is left, and the last one can overshoot.

OR view · Budgeting by the number of runs is like scheduling by job count while ignoring job sizes.

Source: Maia Polo et al. 2024; Zeng and Papailiopoulos 2026; Fisch et al. 2026; Zhou et al. 2025

They budget by *runs*, not *dollars*

The cost depends mostly on the *model*, not the *item*



There is a **62x** difference in median item cost between the cheapest and most expensive of the 33 models.

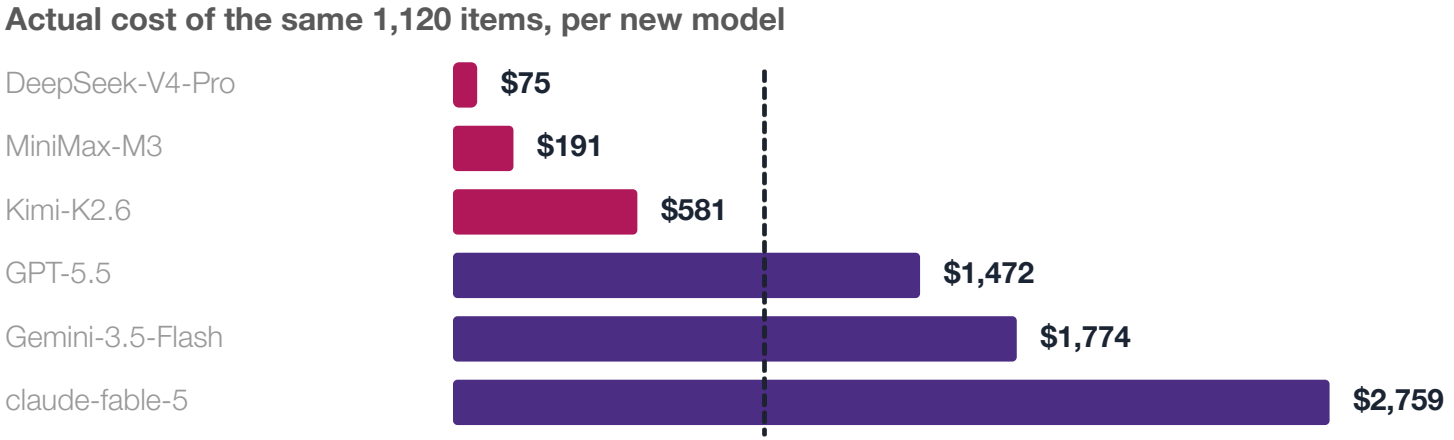
The model explains **58-93%** of the variance in log cost on most benchmarks (36% on GAIA), and the item explains 2-31%.

The six new models fall at very different points in this range.

Data: OpenHands released task costs, SWE-bench with 27 older and 6 newer models

They over rely on historical trends

Older models do not predict the cost of a new model



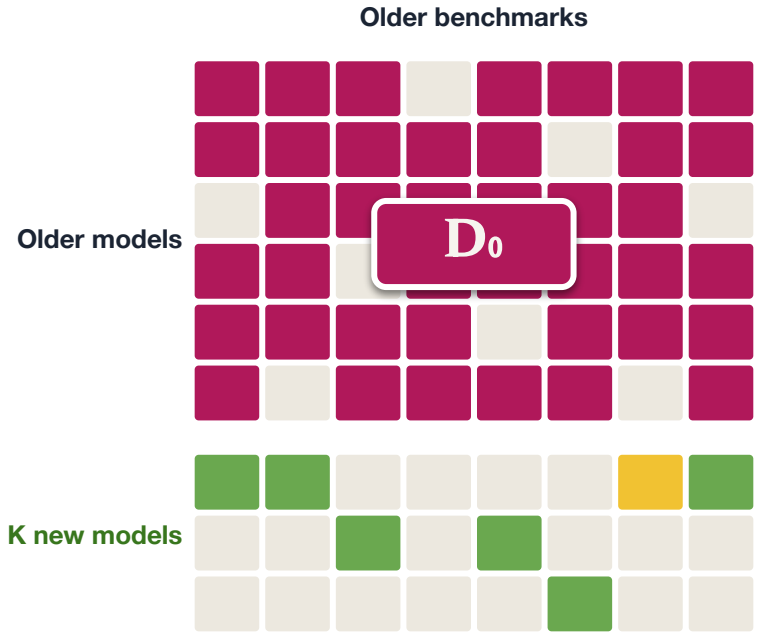
The dashed line is at $981, which is the cost predicted for every model from the 27 older models' per-item averages. Pink = cheaper than predicted, purple = more expensive.

OR view · A per-model multiplier is like learning each server's speed from its first few jobs.

Data: OpenHands released task costs with 27 old models and 6 models released after 21 April 2026. 1,120 shared items on 5 benchmarks.

They ignore budget risk

Expected cost does not capture budget risk

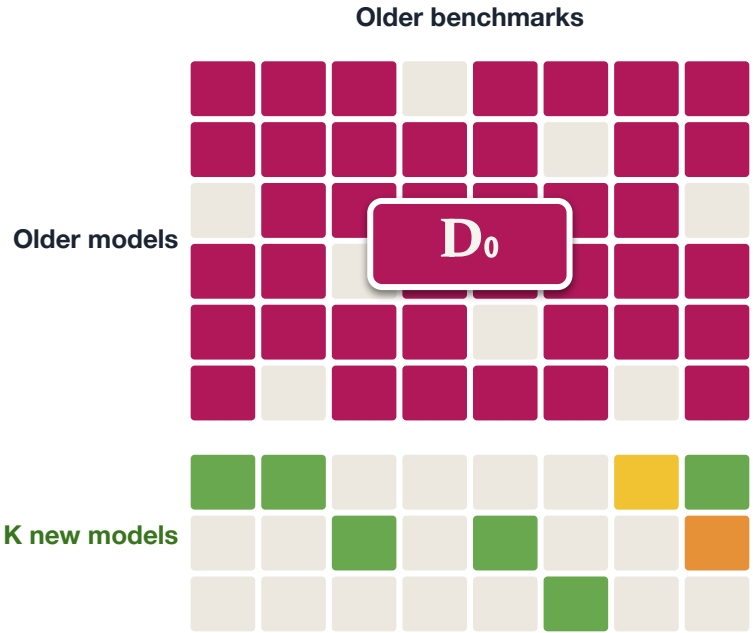


Let's say we have 2 runs

RUN A always cost $40.

They ignore budget risk

Expected cost does not capture budget risk

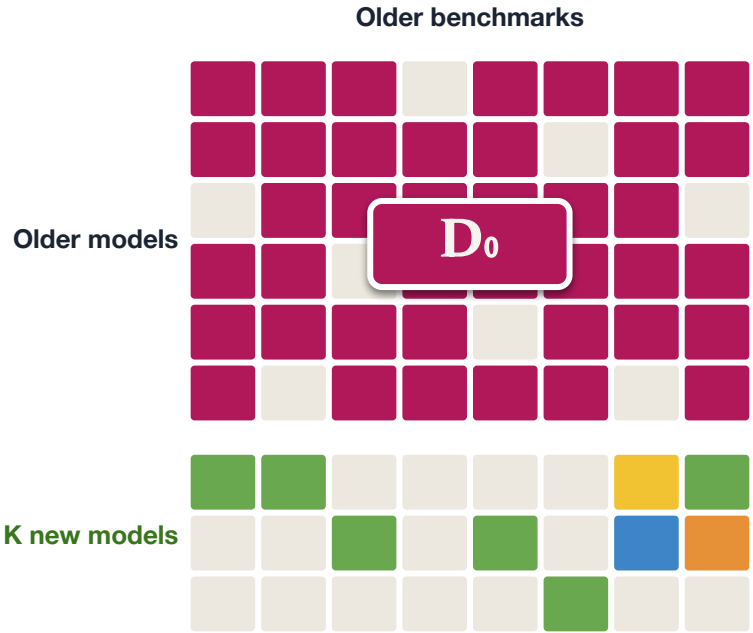


Let's say we have 2 runs

RUN A always cost $40.
RUN B cost $10 80% of the time, or $160 20% of the time.

They ignore budget risk

Expected cost does not capture budget risk



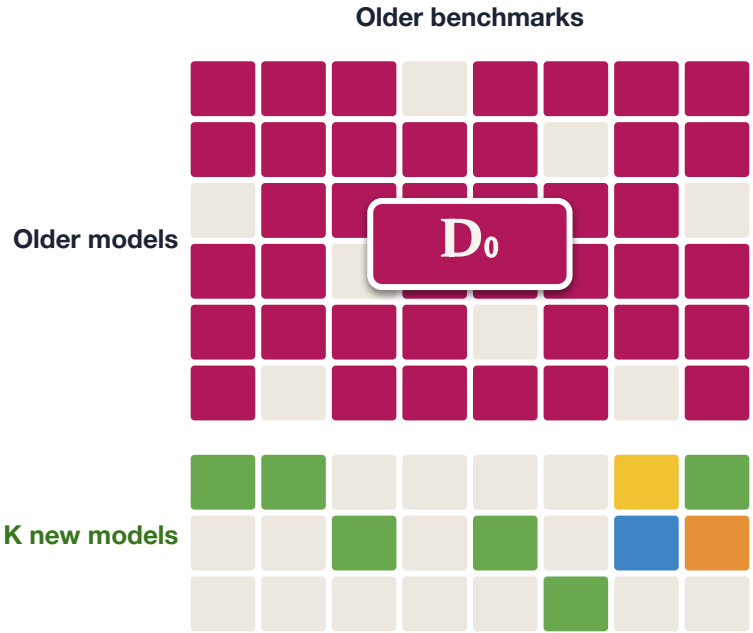
Let's say we have 2 runs

RUN A always cost $40.
RUN B cost $10 80% of the time, or $160 20% of the time.

Let's say we have a subsequent **RUN C** that always cost $60.

They ignore budget risk

Expected cost does not capture budget risk



Let's say we have 2 runs

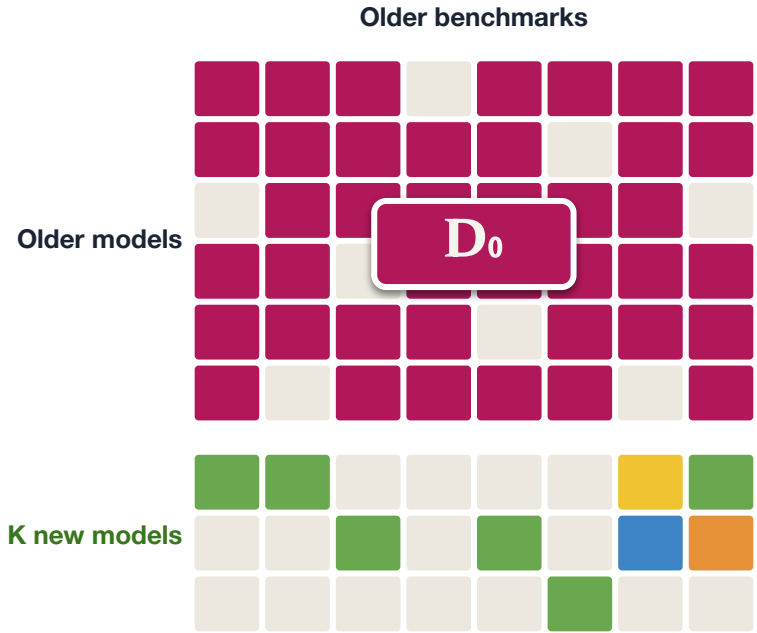
RUN A always cost $40.
RUN B cost $10 80% of the time, or $160 20% of the time.

Let's say we have a subsequent **RUN C** that always cost $60.

When we have $100 left,
choosing **RUN A** always leaves you with an expected number of runs = 2.
choosing **RUN B** leaves you with an expected number of runs = 1.8 (because of 20% overshooting risk).

They ignore budget risk

Expected cost does not capture budget risk



Let's say we have 2 runs

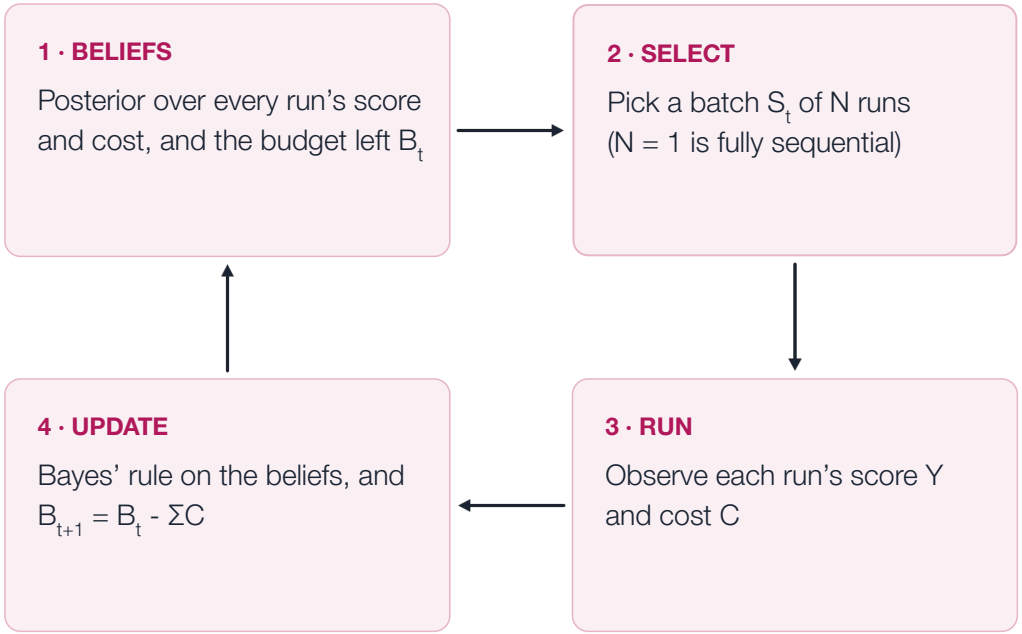
RUN A always cost $40.
RUN B cost $10 80% of the time, or $160 20% of the time.

Let's say we have a subsequent **RUN C** that always cost $60.

When we have $100 left,
choosing **RUN A** always leaves you with an expected number of runs = 2.
choosing **RUN B** leaves you with an expected number of runs = 1.8 (because of 20% overshooting risk).

OR view · Both runs cost $40 on average, but B's cost is far more variable. Queueing theory makes the same point with the Pollaczek-Khinchine formula: mean waiting time grows with the variance of job sizes, not just their mean.

The Bayesian formulation



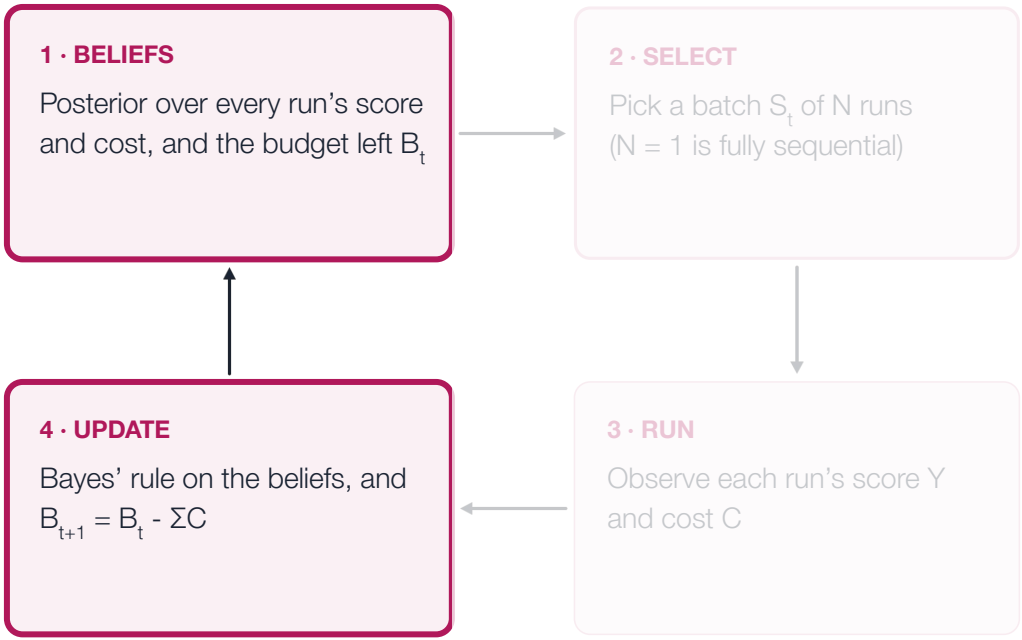
GOAL

Choose the batches to minimize
$E[\Sigma_{kj} (\hat{\theta}_{kj} - \theta_{kj})^2]$
while total cost stays within B

Stop when the budget runs out, then report $\hat{\theta}_{kj}$ for every new model and benchmark.

The hard part is step 2: which N runs?

Steps 1 and 4 need a model of score and cost



Before we 'purchase' a run, what do we expect its score and its cost to be?

Stop when the budget runs out, then report $\hat{\theta}_{kj}$ for every new model and benchmark.

The beliefs are a joint Bayesian model of score and cost

$$p(Y, C \mid a) = p(Y \mid a) \cdot p(C \mid Y, a)$$

PRIOR FROM D_0

$$p_0(\mu, \eta) = p(\mu, \eta \mid D_0)$$

AFTER t RUNS ON THE NEW MODELS

$$p_t(\mu, \eta) \propto p_0(\mu, \eta) \cdot \Pi_t \, p(Y_t, C_t \mid A_t, \mu, \eta)$$

μ is the outcome means and η is the cost parameters. The historical data is the prior. If we have a cold start setting, this prior is wide.

Every run updates beliefs about both scores and costs, and cost may depend on whether the run succeeds or fails.

The estimate is the posterior mean $\hat{\theta}_{kj}$, and its posterior variance is the error we try to reduce.

The outcome model predicts each run's chance of success

$$P(\text{model } k \text{ solves item } i \text{ of benchmark } j) = \sigma(\beta_i + \alpha_{k,j})$$
$$(\alpha_{k,1}, \ldots, \alpha_{k,J}) \sim N(0, \lambda\lambda^T + \Psi)$$

β_i

How easy the item is
Learned from the success rate of older models (fixed)

$\alpha_{k,j}$

How strong the new model is on benchmark j, updated after every run

$\lambda\lambda^T + \Psi$

One shared factor fitted on older models

This links a model's abilities across benchmarks, so a run on one benchmark updates the others

IRT for LLM benchmarks: Maia Polo et al. 2024 · Low-rank completion: Zeng and Papailiopoulos 2026; Fisch et al. 2026

The outcome model predicts each run's chance of success

$$P(\text{model } k \text{ solves item } i \text{ of benchmark } j) = \sigma(\beta_i + \alpha_{kj})$$
$$(\alpha_{k1}, \ldots, \alpha_{kJ}) \sim N(0, \lambda\lambda^T + \Psi)$$

β_i

How easy the item is
Learned from the success rate of older models (fixed)

α_{kj}

How strong the new model is on benchmark j, updated after every run

$\lambda\lambda^T + \Psi$

One shared factor fitted on older models

This links a model's abilities across benchmarks, so a run on one benchmark updates the others

The idea is performance across benchmark is somewhat correlated: if you are good at math, you are likely to be good at coding

IRT for LLM benchmarks: Maia Polo et al. 2024 · Low-rank completion: Zeng and Papailiopoulos 2026; Fisch et al. 2026

New models and benchmarks add new ‘directions’ to the matrix

Older models’ results span a few directions of ability, so the historical matrix is close to low rank.

New benchmarks are built to measure what existing ones miss, and new models bring new capabilities. Each adds a new direction in the latent space.

So a new model has a **familiar part** that history predicts and a **new part** it has never seen.

CONSEQUENCE

A prior fitted to history has no uncertainty in the new direction. It is overconfident, and it undervalues exactly the runs that would reveal the new capability.

Existing algorithms (specifically low-rank bandit algorithms) do not assume this rank growth.

The cost model is log-normal with a learned multiplier per model and benchmark

$$\log C_{ij} = \log \hat{c}_i + a_{mj} + \varepsilon_{ij},\ \varepsilon_{ij} \sim N(0, \sigma^2)$$

$\hat{c}_i$

Expected cost from history: mean cost across older models, or tokens x list price

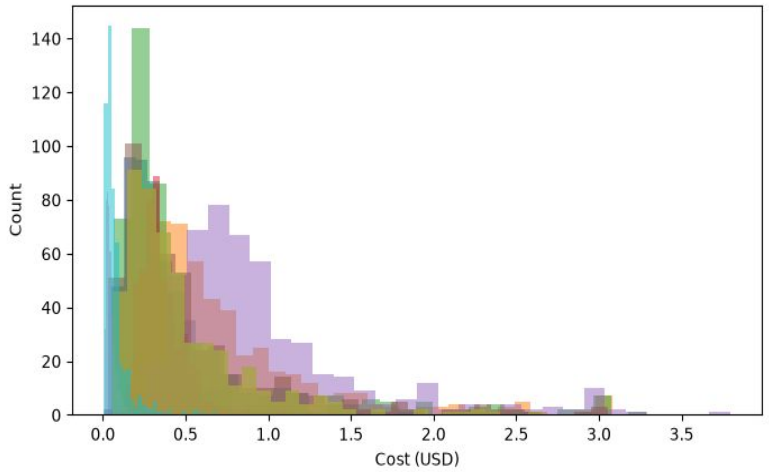
a_{mj}

The new model's multiplier on benchmark j learned online from its first runs

ε_{ij}

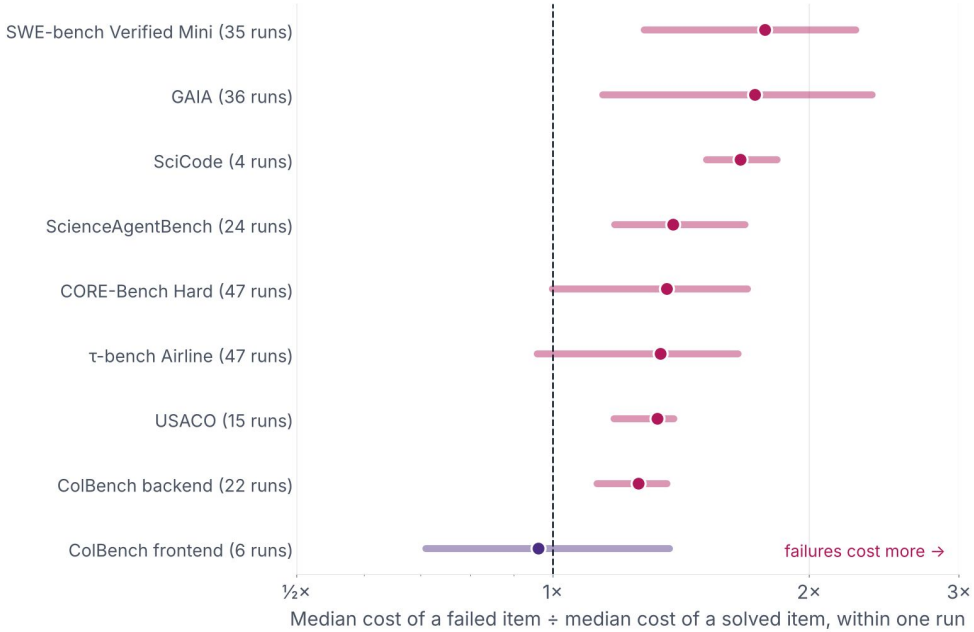
Run-to-run noise Costs are positive and right-skewed

OR view · a_{mj} is the server's speed, learned from its first few jobs. $\hat{c}_i$ is the job's nominal size.



Per-query costs are random and long-tailed: Chen et al. 2026
Data: OpenHands released task costs

Failures usually cost more, so model score and cost jointly



Within a run, failed items cost around **1.3-1.8x** as much as successful ones on 8 of the 9 available benchmarks.

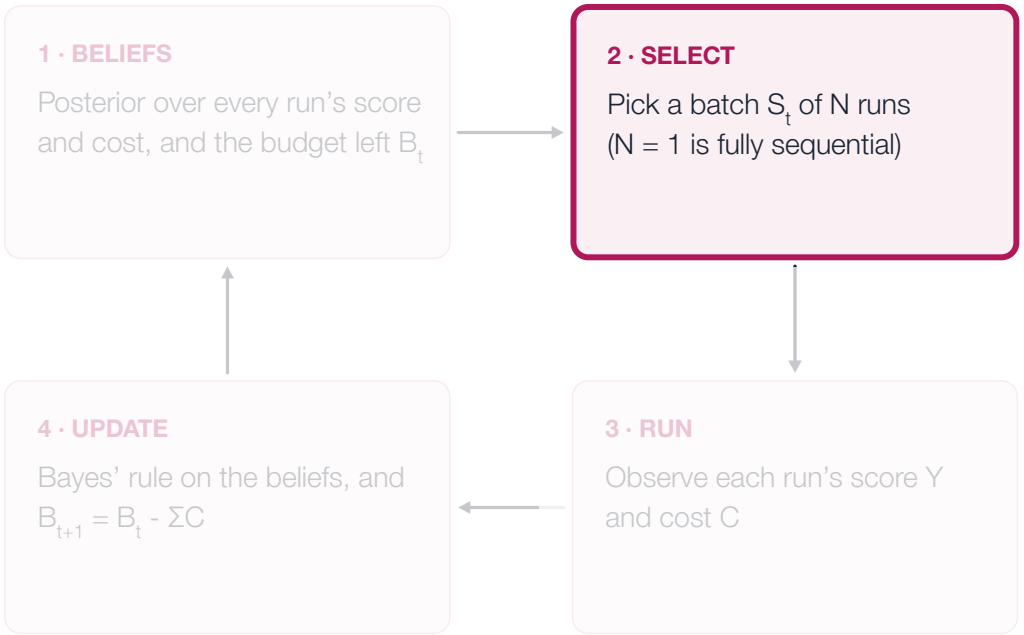
A likely reason is that agents that fail keep trying, which end up using more tokens due to more turns and tool calls.

Our current cost model ignores this. A possible formulation would include a failure term:

$$\log C_i = \log \hat{c}_i + a_{} + \gamma \cdot 1[Y_i = 0] + \varepsilon_i$$

Data: HAL traces, runs with at least 5 solved and 5 failed items; dot = median across runs, bar = middle 50%

Step 2 needs a rule for choosing runs



Given those beliefs and the budget left, which N runs should we 'purchase' next?

Written exactly, this sounds like a Markov decision process (MDP). In practice, we have to approximate it.

The exact problem is an MDP

Part	In our problem
State	Posterior over all scores and costs, plus the budget left, B_t
Action	One run a = (k, j, i) that fits the budget
Transition	The run returns (Y, C) Bayes' rule updates the posterior, and $B_{t+1} = B_t - C$
Reward	At the end, minus the posterior error of the benchmark means

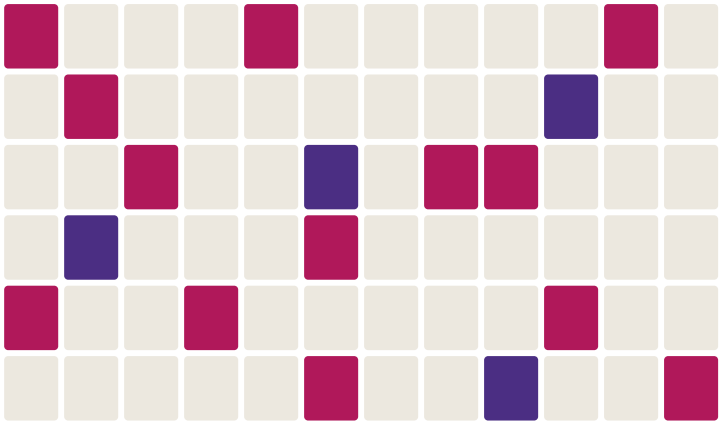
V(p, B) = max over a of
E[V(p updated by (Y, C), B - C)]

WHY WE CANNOT SOLVE IT

The state is a whole distribution, the budget is continuous, and every run branches over outcomes and costs. The tree grows exponentially with the budget.

Source: Budeged Lookahead with Unknown Costs, Astudillo et al. 2021

The exact decision process is far too large



Dark = already run · light = not run
Purple = candidates for the next batch

Actions

6,720 single runs in our OpenHands dataset, which means that batches of 20 give about 10^{58} choices per step.

States

A posterior over 6,720 scores and their costs, plus a continuous budget.

Horizon

Hundreds of runs, and the number depends on costs not yet seen.

Counts from the OpenHands panel: 6 new models x 1,120 items

Why the Gittins index does not apply here

With each (model, benchmark) pair as an arm, the Gittins index would be optimal, but

	Standard bandit	This problem
Goal	Maximize reward, or find the best arm	Estimate every model's score θ_{kj} well
Cost of a pull	Fixed or known	Random, revealed when the run ends. The last run can overshoot the budget.
Arms	Independent	Share low-rank model x benchmark structure, so one run informs other arms
Prior	Describes the arms	New models and benchmarks add directions history has not seen

Each has been studied on its own, but not all four together.

Budgeted bandits: Ding et al. 2013; Xia et al. 2015; Li and Cheung 2024 · Rank growth: thanks to Andrew A. Li

Why the Gittins index does not apply here

With each (model, benchmark) pair as an arm, the Gittins index would be optimal, but

	Standard bandit	This problem
Goal	Maximize reward, or find the best arm	Estimate every model's score θ_{kj} well
Cost of a pull	Fixed or known	Random, revealed when the run ends. The last run can overshoot the budget.
Arms	Independent	Share low-rank model x benchmark structure, so one run informs other arms
Prior	Describes the arms	New models and benchmarks add directions history has not seen

What about a Whittle index? A price per dollar splits the budget into one problem per arm. But the arms still share one model, so a run on one arm changes the others and the problem does not split cleanly.

Gittins 1979 · Whittle 1988 · Budgeted bandits: Ding et al. 2013; Xia et al. 2015; Li and Cheung 2024

Our (current) approximation picks max information/$

1 · LEARN

One model of score and cost

A Bayesian model, started from older models’ results and updated after every run.

Knowledge gradient: Frazier, Powell and Dayanik 2008 · Smith’s rule: Smith 1956 · Static-cost baseline: Krumdick et al. 2026

Our (current) approximation picks max information/$

1 · LEARN

One model of score and cost

A Bayesian model, started from older models' results and updated after every run.

2 · FILTER

A safe set

Drop any run with more than an ε chance of exceeding the remaining budget.

Knowledge gradient: Frazier, Powell and Dayanik 2008 · Smith's rule: Smith 1956 · Static-cost baseline: Krumdick et al. 2026

Our (current) approximation picks max information/$

1 · LEARN

One model of score and cost

A Bayesian model, started from older models' results and updated after every run.

2 · FILTER

A safe set

Drop any run with more than an ε chance of exceeding the remaining budget.

3 · PICK

Information per dollar

Take the N runs with the largest expected drop in uncertainty per expected dollar.

Knowledge gradient: Frazier, Powell and Dayanik 2008 · Smith's rule: Smith 1956 · Static-cost baseline: Krumdick et al. 2026

Our (current) approximation picks max information/$

1 · LEARN

One model of score and cost

A Bayesian model, started from older models' results and updated after every run.

2 · FILTER

A safe set

Drop any run with more than an ε chance of exceeding the remaining budget.

3 · PICK

Information per dollar

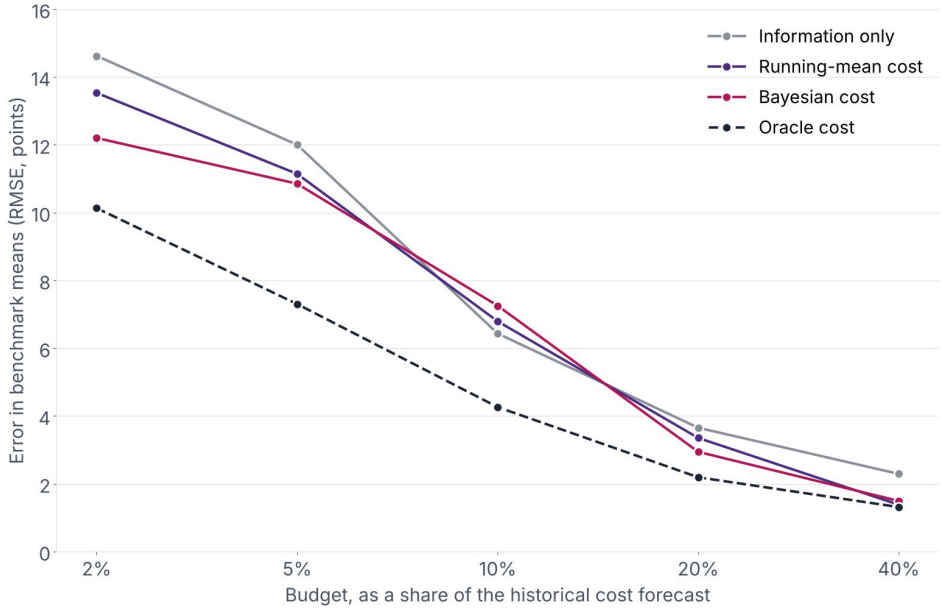
Take the N runs with the largest expected drop in uncertainty per expected dollar.

$$A\square = \text{argmax over safe runs } a \text{ of } \Delta(a) / E[C \mid a]$$

This is a heuristic, not the optimal policy. It looks one step ahead, so it undervalues runs that pay off later, such as learning a new model's cost level. In OR terms it is a **knowledge-gradient rule per unit cost**, or Smith's rule with job sizes learned online. The safe set is **admission control**.

Knowledge gradient: Frazier, Powell and Dayanik 2008 · Smith's rule: Smith 1956 · Static-cost baseline: Krumdick et al. 2026

Preliminary results show cost awareness helps most at tight budgets



Information only ignores cost and the oracle knows every run's true cost.

At a 2% budget, learning costs cuts error from 14.6 to 12.2 points.

The rule is not always better: at 10% it loses to ignoring cost.

Knowing the true costs (dashed) would do far better, so a better policy has room to gain.

OR view · The gap to the oracle is the value of knowing job sizes in advance.

Open problems in this project

Problem	Closest OR idea
An index policy when arms share structure and the objective does not add up across arms	Gittins and Whittle indices
A prior that expects each new model or benchmark to add a direction	Learning with a growing number of factors
How much an adaptive plan beats a fixed one	The adaptivity gap in stochastic knapsack
When to cap a run and keep a partial result	Preemption, FB scheduling, decreasing failure rate
Bounding overrun over the whole evaluation, not one step at a time	Chance constraints, CVaR

Other LLM problems with OR analogs

LLM problem	OR analog
Routing: send each query to one of several models with different price, speed and quality	Task assignment in a server farm of unlike servers; contextual bandits
Serving: order requests whose output length is unknown until generation ends	Scheduling with unknown job sizes (SRPT, Gittins)
Human checking: spend a small budget of human labels where the LLM judge is most likely wrong	Budgeted inspection and sampling
Reliability: decide how many repeat runs an agent needs on each item	Sequential sampling, optimal stopping
Model selection: find the best model, or the top k, without scoring them all	Ranking and selection, best-arm identification

Thank you!

References

- Astudillo, Jiang, Balandat, Bakshy and Frazier. Multi-step budgeted Bayesian optimization with unknown evaluation costs. NeurIPS 2021.
- Carpentier, Munos and Antos. Adaptive strategy for stratified Monte Carlo sampling. JMLR 16(68), 2015.
- Ding, Qin, Zhang and Liu. Multi-armed bandit with budget constraint and variable costs. AAAI 2013.
- Cordero Encinar, Cemgil, Doucet, Aglietti and Chiappa. BayesAME: Bayesian active model evaluation. arXiv:2607.27023, 2026.
- Fisch et al. CollabEval: statistically efficient collaborative model evaluation via matrix completion. arXiv:2607.05046, 2026.
- Ghosh, Mai, Channing and Choshen. AI evals are becoming the new compute bottleneck. EvalEval Coalition blog, April 2026.
- Kapoor et al. Holistic agent leaderboard: the missing infrastructure for AI agent evaluation. ICLR 2026.
- Krumdick, Wiemerslage, Ebner, Lovering and Tanner. Cost-efficient estimation of general abilities across benchmarks. arXiv:2604.01418, 2026.
- Li, Ma, Ballesteros, Benajiba and Horwood. Active evaluation acquisition for efficient LLM benchmarking. ICML 2025.
- Li and Cheung. Best arm identification with resource constraints. AISTATS 2024.
- Maia Polo, Weber, Choshen, Sun, Xu and Yurochkin. tinyBenchmarks: evaluating LLMs with fewer examples. ICML 2024.
- Woodroofe and Hardwick. Sequential allocation for an estimation problem with ethical costs. Annals of Statistics 18(3), 1990.
- Wu, Nair and Candès. Efficient evaluation of LLM performance with statistical guarantees. arXiv:2601.20251, 2026.
- Xia, Li, Qin, Yu and Liu. Thompson sampling for budgeted multi-armed bandits. IJCAI 2015.
- Yuan et al. Beyond one-size-fits-all: tailored benchmarks for efficient evaluation. ACL 2025.
- Zeng and Papailiopoulos. You don't need to run every eval. arXiv:2606.24020, 2026.
- Zhang, Dorner and Hardt. How benchmark prediction from fewer data misses the mark. NeurIPS 2025 workshop.
- Zhou, Belardi, Wu, Zhang, Gomes, Sun and Weinberger. On speeding up language model evaluation. ICLR 2025.
- Chen, Zhang, He, Stoica, Zaharia and Zou. The price reversal phenomenon: when cheaper reasoning models end up costing more. arXiv:2603.23971, 2026.
- Dean, Goemans and Vondrák. Approximating the stochastic knapsack problem: the benefit of adaptivity. Mathematics of Operations Research, 2008.
- Frazier, Powell and Dayanik. A knowledge-gradient policy for sequential information collection. SIAM Journal on Control and Optimization, 2008.
- Gittins. Bandit processes and dynamic allocation indices. Journal of the Royal Statistical Society B, 1979.
- Gupta, Krishnaswamy, Molinaro and Ravi. Approximation algorithms for correlated knapsacks and non-martingale bandits. FOCS 2011.
- Harchol-Balter. Performance Modeling and Design of Computer Systems: Queueing Theory in Action. Cambridge University Press, 2013.
- Rockafellar and Uryasev. Optimization of conditional value-at-risk. Journal of Risk, 2000.
- Ruan, Maddison and Hashimoto. Observational scaling laws and the predictability of language model performance. NeurIPS 2024.
- Smith. Various optimizers for single-stage production. Naval Research Logistics Quarterly, 1956.
- Whittle. Restless bandits: activity allocation in a changing world. Journal of Applied Probability, 1988.