The Allocation of Machine Error: Where AI Errors Go When Models Improve

This draft: October 30, 2026

Abstract

Organizations adopting AI must learn not only which tasks a model performs well, but also where it is likely to fail. Yet this knowledge may become unreliable as LLM developers continually update their models. We study how LLM development changes the distribution of errors across firms and tasks. Theories of human heuristics explain how decision makers with limited capacity rely on simplifying rules that produce systematic errors. LLMs also make systematic errors, but their patterns shift with the priorities of external developers, which are often undisclosed and change frequently. When capacity is binding, reducing errors in one area increases errors elsewhere, most for firms whose needs receive the least weight in development. Even prioritized firms may lose, because improvements on some of their tasks can shift errors to others and make their knowledge of where the model fails obsolete. A shift in priority lowers the total cost of errors only when it aligns priorities with what errors cost. We test our predictions in experiments with LLMs and find that repeatedly targeting one firm's failures increases errors for other firms and can increase errors on the prioritized firm's other tasks. Firms' knowledge of AI failures thus has an uncertain shelf life, and the costs of model errors depend on whose needs developers prioritize.

Keywords: artificial intelligence, bounded rationality, machine heuristics, error allocation, AI evaluation, model updates, organizational knowledge.

JEL: D83, D91, D62, L15, L86, M15, O33.

Financial support: None.
Conflict of interest: None.
AI disclosure: Claude models (Anthropic), most recently Claude Opus 5.5, were used to assist in deriving the proofs and verifying them numerically. They were also used for writing assistance. Everything has been verified, and all errors remain our own.

1 Introduction

Each new release of a large language model (LLM) promises improvements in performance, yet these improvements can come with new errors. Existing studies document various instances in which larger or newer models fail on inputs that smaller or earlier models handled correctly (Zhong et al., 2021; Echterhoff et al., 2024), and sometimes even within the same model family over just three months (Chen et al., 2024). Where a model improves and where it fails ultimately depend on what the model developers prioritize during their development. These priorities are often opaque to users and may change over time (Wan et al., 2025; Ott et al., 2022; Kiela et al., 2021).[1]

These changes in where the errors go have important implications for firms deciding where and how to adopt AI. Beyond a model's overall accuracy, firms need to understand which tasks it performs well and where it fails so that they can incorporate AI effectively in places where it does perform well. Yet this knowledge may become obsolete as developers update their models, even when the overall performance of the model improves. This paper asks how LLM development reallocates the errors across firms and tasks, who bears the costs, and how these outcomes change with developers' changing priorities.

To answer these questions, we build on the theories of bounded rationality. The seminal work of Simon (1955) argues that rational behavior is still bounded by the limited information and computing capacity that decision makers usually have, which leads to heuristics and behaviors that may not appear to be fully rational. A large language model is a decision maker of this kind, with its capacity limited by its model size, architecture, and its training, and thus it is bound to get some inputs wrong in a systematic pattern, just as humans do (Han, 2025). Researchers infer a person's heuristics from the behavior and decisions that they produce (Tversky and Kahneman, 1974), and experiments can change the environment to

[1]For example, in April 2025, OpenAI rolled back its update to GPT-4o after complaints from users that the updated model was flattering users and agreeing too much (OpenAI, 2025b). Later OpenAI acknowledged that the issue came from its decision to put more weight on users' thumbs-up and thumbs-down feedback during training, and that it was not adequately disclosed in the release notes (OpenAI, 2025a).

examine how humans adopt heuristics that suit the environment (Gigerenzer and Brighton, 2009), but no experimenter or external party can set the priorities inside that process in which heuristics arise. In contrast, machine heuristics are directly a consequence of the training process whose priorities are determined by its developer. We refer to this setting as artificial bounded rationality. Because model training improves on errors where the developer prioritizes and gives more weight in its objective function, the errors that remain accrue to the areas that are weighted the least by the developer. Therefore, the errors are not random, but are predictable and systematic.

We study this in a simple setting. An LLM performs tasks for firms, such as reviewing contracts or answering customer questions. The output of the model is either correct or incorrect, which is useful for firms in deciding where to deploy AI. For each task, there are different inputs (e.g., different contracts), and error rates differ across tasks. The developer has a fixed capacity and allocates its capacity across tasks. The failure rate of each task is then a function of how much capacity it receives. The developer divides its capacity to minimize the error rates weighted by its priorities. The priorities may be determined by observing benchmarks, user reports, and the potential revenues and risk associated with each task, though we are agnostic about how exactly these priorities are set. A firm learns where the model fails from its experience, and each firm keeps a list of inputs on which the model fails (e.g., a particular contract that the model misread), and a map of which of its tasks the model fails on more often (e.g., the model makes more errors on hard contracts than on routine ones). We illustrate our results in the simple case of a firm that uses the model for two tasks, reviewing routine contracts and reviewing hard contracts. A competitor uses the model for the same two tasks, and other unrelated firms use the model for other tasks.

We then examine where the residual errors go and how they affect firms' existing knowledge. The developer allocates capacity to where failures have the highest priorities, so a task's error rate falls as its priority rises, and in our model they are inversely proportional.

The tasks with low priority do not receive any capacity, and the model fails on those tasks. When a model has spare capacity, a developer can lower the error rate on some tasks without raising the errors on others. However, when the model is at the frontier of its capacity, lowering the error rate on the tasks that gain priority raises the error rates on all other tasks the model serves, and, in absolute terms, most on those with the lowest priority. Thus a new release of the model improves performance on the tasks that the developer prioritizes, but performs worse on the ones that did not receive priority. Importantly, what determines whether or not the model is at its capacity is not the absolute computing power or model size, but how demanding the work it is given is. Therefore, this is not a problem that becomes relevant only in the future, when LLMs run out of capacity. For every model, there will be work complex enough that the problem of error allocation will arise.

How changes in the error allocation affect firms then depends on who receives the priority in the development. When the developer prioritizes the tasks that other unrelated firms use, the error rates for a given firm rise in the same proportion on both its routine and hard contracts, so its overall error rate goes up, which is costly, but its map remains correct. On the other hand, when the developer prioritizes the task of a competitor with overlapping tasks, say the hard contracts, the error rate on the hard contracts falls but the error rate on the routine contracts goes up. If the hard contracts become less error-prone than the routine ones, the firm's map of where it can use AI becomes obsolete, which may outweigh the benefit of the lower error rate on the hard contracts.

Whether the change in priority is good for the users as a whole depends on whether the developer priorities already reflect the total user costs in aggregate. If they do, then raising the priority of certain tasks at capacity will raise the total costs of errors. A change in priority can lower cost only when the priority was not aligned with the user costs.

We test these predictions with large language models. We use the Pythia models with 70, 160, and 410 million parameters, and Qwen3, an open model from Alibaba with 0.6 billion parameters (Yang et al., 2025). Pythia is a family of open-source large language models that

have been developed for scientific research such that all models have been trained on the same texts and the model sizes differ mainly in their number of parameters (Biderman et al., 2023). We then fine-tune each model to perform eight different text classification tasks, such as judging the sentiment of a movie review or identifying the topic of a news article. We mimic the developer's priorities by giving each of the eight tasks a different weight in training. A focal firm and its competitor use the same two tasks, and an unrelated firm uses two other tasks. In each run, the tasks for the firms are randomly chosen, such that the differences in the results are not driven by the differences in the inherent difficulty of the tasks. The focal firm has seen half of its inputs, and keeps a list of the inputs the model got wrong (e.g., the particular news articles whose topic the model misidentified), and uses that knowledge to fit a map that predicts which of its other inputs the model is likely to get wrong (e.g., that articles covering both business and technology are more likely to be misidentified). When the focal firm reports the errors, the developer adds the reported inputs, with their correct answers, to the training data at four times the normal weight and continues training the model. We compare this with a model update without a report, and with a report from the competitor. We also run six rounds of reports. The models differ in how well they fit their training data. In our setting, the two smaller Pythia models and Qwen3 with only its top layer trained (to mimic a model with less capacity while keeping the same pretrained model) get about a fifth to a third of their training inputs wrong. Meanwhile, the largest Pythia model and Qwen3 with at least half of its layers trained get fewer than one in ten wrong.

In every model that we examine, the error rate falls on the tasks that receive more priority in training. However, a report also creates new errors for the focal firm. After a single report, the share of the focal firm's not-yet-seen inputs that switch from right to wrong is 5.6 to 12.1 percentage points higher than after a model update without a report. A competitor's report on the same tasks has nearly the same effect. When models fit the training data poorly, six rounds of reports remove about an eighth to a fifth of the predictive power of the

firm's original map, suggesting that the firm's knowledge of where the model fails may suffer from the model update. In addition, the model update also costs the unrelated firm that never reports. In the models that fit poorly, after six rounds of reports and updates, the unrelated firm's error rate rises by 1.1 to 2.2 percentage points compared to the case without any reports, and the rise occurs in 43 out of 50 runs.[2] On the other hand, in models that fit their training data well, an error report fixes the reported errors and raises the unrelated firm's error rate by only about half a percentage point.

These results imply that knowledge about a model has a shelf life that can expire with a model update, and that how long that knowledge lasts is set by the developer's priorities in updating the model. In our setting, a model update affects the firm's existing knowledge of where the model fails in two different ways. When the update adds model capacity, the error rates fall in the same proportion, so the firm's existing knowledge of where the model fails stays intact. On the other hand, when the update reflects changes in the developer's priorities, the model can reallocate where the error occurs, which can degrade the firm's knowledge. Because the developer's priorities are often undisclosed and change often, a model improvement is not always good news for firms but a risk to monitor, and an improvement may benefit firms with high priority at the expense of those without it. Monitoring how developers' priorities may change, such as through changes in ownership, investments in the developer by a firm's competitors, and public offerings, can therefore be an important part of managing that risk.

The paper is related to two strands of literature. First, the paper builds on work on bounded rationality, which arises when a decision maker with limited capacity must solve complex problems (Rubinstein, 1986; Salant, 2011; Gabaix, 2014). We view AI models in this setting, given their universal applications across many complex tasks while their capacity is finite. However, unlike human heuristics, an LLM's simplifications are chosen by its

[2]An error report adds new examples to the training data as well as giving them more weight. To separate the two effects, we compare runs that add the same examples but differ only in their training weights. We find that the differences in the training weights alone accounts for most of the increase in the error rate of unrelated firms.

developer, whose priorities need not match the costs of the firms that potentially bear the model's errors. Second, the paper contributes to the growing work on AI in roganizations. Recent papers show that newer AI models perform well on some tasks but poorly on others compared to their older counterparts, and their reported bencharks can mispresent their performance (Dell'Acqua et al., 2026; Gans, 2026). Likewise, a developer that trains on what is evaluated can neglect those that are not, similar to the idea of the multitasking models (Zhuang and Hadfield-Menell, 2020; Wang and Huang, 2026; Truong et al., 2026). We add to this literature by showing why do these uintended errorse arise systematically, and how the developer priorities ultimately determine where the errors occur and who ultimately pays for the osts when the models improve.

2 Related Literature

3 Model

We first describe our model, in which a developer trains a model with limited capacity and firms use it. The developer may be a frontier lab such as OpenAI or Anthropic, or a company that trains its own model.

3.1 Inputs, capacity, and priorities

A developer trains a model on different types of inputs $j = 1, \ldots, J$. A type is a kind of input that is used for different tasks, and which the developer can tell apart from others (e.g., customer question versus a sales contract). In the introduction, we call each type a task. Type j's share of all inputs is $\mu_j > 0$, and $\sum_j \mu_j = 1$. In our simple example, there are four types of inputs: benchmark, routine, hard, and niche. The benchmark type is the input that public evaluations test heavily. The routine and hard types are the inputs of a customer firm, which we call the focal firm. The model fails more often on the hard type

than on the routine type. The niche type is used by other firms, and the developer rarely sees it. The shares of the benchmark, routine, hard, and niche types are 0.4, 0.2, 0.2, and 0.2.

The developer has capacity $C > 0$ to spend on training its model. It spends $k_j \geq 0$ on each type subject to $\sum_j \mu_j k_j \leq C$, and the allocation of capacity determines the error rate on each type:

$$e_j = e^{-k_j}. \tag{1}$$

Thus, each additional unit of capacity cuts a type's error rate by the same proportion. Because e^{-k} is positive for every finite k, every type keeps some error as long as capacity is finite, which is the premise of limited computational capacity in Simon (1955).

The developer does not weigh errors equally. It assigns priority $v_j > 0$ to type j, which is how much a failure on type j counts. Every type has some priority, because the pretraining objective counts failures on every type. The developer then allocates its capacity to minimize its priority-weighted error:

$$\min_{k_1,\ldots,k_J \geq 0} \sum_{j=1}^{J} \mu_j v_j e^{-k_j} \qquad \text{subject to} \qquad \sum_{j=1}^{J} \mu_j k_j \leq C. \tag{2}$$

The error rate of type j thus enters the objective with weight $\mu_j v_j$, its share times its priority. We refer to the vector of error rates $(e_1, \ldots, e_J)$ as the model's error allocation. In our example, we set the priorities of the benchmark, routine, hard, and niche types to 8, 2, 1, and 1/8, and capacity to $C = 3 \ln 2 \approx 2.08$.

We take the priorities as given and remain agnostic about how they are set. In reality, they are likely driven by both measurement and preferences. Measurement determines which failures the developer sees, which depends in part on which firms report their failures. If a developer rarely sees failures on a type, like the niche type, it gives that type a low priority. On the other hand, not all visible failures receive the same priority. For example, types that generate more revenue are likely to receive greater priority. Our results on the model's error

rate, the total share of inputs that fail, weight every failure equally, so they measure a social loss only to the extent that the priorities differ from the social cost of each failure. Unlike in Wilder et al. (2020) and Bryan and Gans (2026), the developer's priorities do not depend on whether firms check particular outputs, and the developer learns which failures a firm has found only when the firm reports them.

Definition 1 (Machine heuristic)**.** Given the developer's priorities v_j and capacity C, the machine heuristic is the error allocation that solves problem (2).

By design, the developer leaves the most error on the types with the lowest priority, so the errors of the machine heuristic can be predicted from the type of the input.

3.2 Firms, lists, and maps

Firm i's share of usage in each type j is w_{ij}, with $\sum_j w_{ij} = 1$, and the error rate on its inputs is $\sum_j w_{ij} e_j$. For our simplified case, the focal firm splits its usage between the routine type and the hard type by $w_{\text{routine}} = 0.5$ and $w_{\text{hard}} = 0.5$. A competitor uses the same two types, and the other unrelated firms use the benchmark type or the niche type.

A firm keeps the knowledge of where the model fails, which is an important step in deciding where and whether to incorporate AI into its organization. First, the firm maintains a list of inputs on which the model has failed (e.g., a specific contract that the model got wrong), which the firm checks before relying on the model's output. Second, the firm uses its experience to form a map of where the model generally fails, which ranks the types it uses by their error rates (e.g., the model performs more poorly on hard contracts than on routine contracts). The firm uses its map to decide which types of input to check beyond its list, which covers only the inputs it has already seen fail. A map is correct when it ranks the types as their current error rates do, with ties where the rates are equal. A map goes stale when it ranks a type above another type that now has a strictly higher error rate.

A model error costs the firm c for each error it does not verify, and $c_h < c$ for each error it verifies, such that a caught error is cheaper than an uncaught error. If the input is on

the list, then the firm verifies the error. We define a_i as the share of firm i's inputs that are listed failures, and b_{ij} as the share that are unlisted failures for each type j. Apart from its listed failures, the inputs the firm uses are inputs it has not yet seen, so a given input that is not on the list fails at its error rate e_j, so b_{ij} moves in direct proportion to e_j when the error rates change. Beyond its list, the firm checks a share s of its inputs from the type that its map ranks as the most error-prone, which we denote j_1 (either one if the map ranks two types equally first), and verifies the errors it finds there. The firm checks only the share s because checking every output takes time and resources.[3] We assume that s is small enough that these inputs are not already on the list, for whichever type the firm checks, and they fail at e_{j_1}. The firm's total cost of error is then

$$c \sum_{j=1}^{J} b_{ij} + c_h \, a_i - (c - c_h) \, s \, e_{j_1}, \tag{3}$$

where the last term corresponds to the savings from catching the error, which is a function of its map. When the map no longer ranks the most error-prone type first, the firm checks a type with a lower error rate rather than the one with the highest error rate, and therefore, its cost will increase by

$$(c - c_h) \, s \left(\max_{j:\, w_{ij} > 0} e_j - e_{j_1} \right).$$

3.3 Model updates and reports

The developer updates the model when its priorities change, for example after it receives a report of model failures from a customer. The update then changes the error rate on each type j from e_j to e'_j. Given the random nature of the training process for LLMs, even if the error rate remains unchanged, $e_j = e'_j$, the specific inputs that fail may not be the same. For example, 25 out of 100 contracts can fail in both models, but a completely different set

[3] We fix s for simplicity, but a natural extension would be to introduce the cost of checking $\kappa > 0$ so that the firm checks inputs in the order of its map for as long as the expected saving from a check, $(c - c_h)e_j$, exceeds κ.

of 25 contracts may fail while the original 25 succeed in the new model. Thus, whether the inputs that failed in the previous model still fail depends on how much the updated model reshuffles the model failures. We model this by assuming that with probability $1 - \chi$ the update keeps the input's past outcome as far as the new error rate allows. Thus, a contract that the model read correctly remains correct unless an increase in the error rate breaks it, and a contract that the model misread remains misread unless a drop in the error rate fixes it. With probability χ, the update redraws the input so that the past outcome is not informative about the new outcome. Either way, the input fails at the new rate e'_j, so churn changes which inputs fail but not how many. Precisely, a kept input that failed before still fails with probability $\min\{1, e'_j/e_j\}$, a kept input that succeeded before fails with probability $(e'_j - e_j)^+/(1 - e_j)$, and inputs fail independently of each other. We refer to $\chi \in [0, 1]$ as the churn rate.

A firm can report model failures to the developer. Let S denote the set of types that are contained in a firm's report. For our simplified setting, the focal firm's report can contain $S = \{\text{routine}\}$, $\{\text{hard}\}$, or $\{\text{routine}, \text{hard}\}$. Because the focal firm checks only its hard contracts, we take $S = \{\text{hard}\}$. If the developer acts on the firm's report, it fixes the reported inputs, so that the reported inputs get fixed, multiplies the priority of each type in S by $1+\beta$, with $\beta > 0$, and solves problem (2) again at the same capacity. For our simplified case, we assume $\beta = 3$, so the priority of the reported type is multiplied by four.

Table 1 summarizes the parameters and results of our simplified model.

Table 1: The four-type example

	Benchmark	Routine	Hard	Niche
Panel A: Parameters and allocation				
Share of inputs, μ_j	0.4	0.2	0.2	0.2
Priority, v_j	8	2	1	1/8
Capacity allocated, k_j	3.47	2.08	1.39	0
Error rate, e_j (%)	3.1	12.5	25.0	100.0
Panel B: Error rate after a report (%)				
Report on the hard type	4.4	17.7	8.8	100.0
Report on the benchmark type	1.6	25.0	50.0	100.0

This table reports the assumed parameters and error rates for each of the four types (benchmark, routine, hard, and niche) in our simplified case. Panel A reports the parameters, the capacity allocated to each type, and initial error rates. Panel B reports the error rates after a report on the hard type and after a report on the benchmark type. Capacity is $C = 3\ln 2 \approx 2.08$. See Section 3 for details on our model setup.

4 Allocation of Machine Error

We now derive the results on where the developer leaves errors and how this allocation of errors responds to changes in the developer's priorities. Proofs are in Appendix A.

4.1 Where the residual errors go

We first solve the developer's problem (2), in which the developer allocates a fixed capacity across different input types to minimize the priority-weighted model errors. A type is abandoned when it receives no capacity, and served when it receives some.

Proposition 1 (Allocation)**.** *The error rates that solve problem* (2) *are*

$$e_j = \min\left\{1,\ \frac{\lambda}{v_j}\right\}, \tag{4}$$

where $\lambda > 0$ *is the unique solution of*

$$\sum_{j=1}^{J} \mu_j \big(\ln v_j - \ln \lambda\big)^+ = C. \tag{5}$$

Types with $v_j > \lambda$ are served, and types with $v_j \leq \lambda$ are abandoned and have error rate one. If every type is served, then $\lambda = e^{-C}G$ and $e_j = e^{-C}G/v_j$, where $G = \exp\left(\sum_j \mu_j \ln v_j\right)$ is the geometric mean of the priorities, weighted by the type shares.

The proof follows from the first-order condition $v_j e^{-k_j} = \lambda$ for a served type. The developer adds capacity to a type up to the point where its priority-weighted error equals the common level λ, and thus equates the priority-weighted error across all served types. The types whose priority is at or below λ are abandoned.

The error on a served type is therefore inversely proportional to its priority. In the simplified model with four types in Table 1, $\lambda = 1/4$, and the error rates on the benchmark, routine, and hard types are 1/32, 1/8, and 1/4, respectively. The niche type, on the other hand, has priority $1/8 < \lambda$, and is thus abandoned.

Unequal priorities also raise the model's error rate. The model's error rate is lowest when capacity is distributed evenly across all types, which is what a developer with equal priorities does. Each unit of capacity reduces a type's error by the same proportion, and therefore removes the most errors where error is highest. With unequal priorities, capacity is allocated toward the highest-priority types, where error rates are already low, so each unit moved there removes fewer errors than it adds on the types it is taken from. For example, with two equally common types with priorities 4 and 1, respectively, the developer leaves an error rate 25 percent higher than with equal priorities.[4]

4.2 Compensation and relocation

We next examine what happens when the developer changes priorities without adding more capacity. For simplicity, we assume that the developer raises the priority of the types that firms report to have errors, though we remain open to other exogenous reasons why the developer may shift its priorities in training for the next release. Let S be the nonempty set

[4]By Proposition 1, $k_1 = \ln(4/\lambda)$ and $k_2 = \ln(1/\lambda)$. The capacity constraint $\frac{1}{2}k_1 + \frac{1}{2}k_2 = C$ gives $\lambda = 2e^{-C}$, so the model's error rate is $\frac{1}{2}(\lambda/4 + \lambda) = \frac{5}{4}e^{-C}$. With equal priorities, $k_1 = k_2 = \ln(1/\lambda) = C$, so $\lambda = e^{-C}$ and the model's error rate is e^{-C}. Both types are served when $C > \ln 2$.

of reported types, and suppose every type in S is served. Let m be the share of the inputs of served types that belong to S, $m = \sum_{j \in S} \mu_j \big/ \sum_{j:\, v_j > \lambda} \mu_j$, and let e'_j be the error rate in the new model after the update. The condition $m < 1$ means that some served types lie outside S.

Proposition 2 (Compensation and relocation)**.** *Suppose that, holding capacity fixed, the developer raises the priority of the reported types in S by the factor $1 + \beta$, with $\beta > 0$ and $m < 1$, and that $(1 + \beta)^m e_l < 1$ for every served type l outside S, so that the change in priority does not raise the error rate of any served type to one, which holds whenever β is small enough.*

(i) (Relocation.) The error rate of every type in S is multiplied by $(1 + \beta)^{m-1} < 1$, the error rate of every served type outside S is multiplied by $(1 + \beta)^m > 1$, and abandoned types' error rates remain equal to one.

(ii) (Compensation.) Measured with the old priorities, the error removed from S is at most the error added outside S,

$$\sum_{j \in S} \mu_j v_j \, (e_j - e'_j) \;\leq\; \sum_{j \notin S} \mu_j v_j \, (e'_j - e_j). \tag{6}$$

The outline of the proof is as follows, where we treat λ as the price of capacity. From Proposition 1, a served type is allocated capacity $k_j = \ln v_j - \ln \lambda$, so that its priority-weighted error $v_j e_j$ equals λ. If its priority rises by the factor $1 + \beta$, its error rate must fall by the same factor to bring $v_j e_j$ back to λ, and at the old price this takes $\ln(1 + \beta)$ more capacity. The types in S would therefore demand $\mu(S) \ln(1 + \beta)$ more capacity in total. Because capacity is fixed, the price of capacity, λ, must rise until this demand is met by a reduction in the capacity of all served types. Because a served type's capacity is $\ln v_j - \ln \lambda$, raising $\ln \lambda$ by Δ takes Δ from every served type and frees $\mu_I \Delta$ in total, where μ_I is the share of all served types. Equating the additional capacity demanded to the capacity freed

gives $\mu(S)\ln(1+\beta) = \mu_I \Delta$, or $\Delta = m\ln(1+\beta)$, so the price rises by the factor $(1+\beta)^m$.[5] Because $e_j = \lambda/v_j$, the error rate of every served type is multiplied by $(1+\beta)^m$. For every served type not in S, the error rate therefore rises by this factor, while for those in S, whose priority also rose by $1+\beta$, the error rate is multiplied by $(1+\beta)^m/(1+\beta) = (1+\beta)^{m-1} < 1$. For abandoned types, the error rate remains one. The types in S stay served, because they were served before and their priority rises faster than the price, and the condition on e_l keeps every other served type served.

The new allocation e'_j cannot lower the priority-weighted error under the old priorities, since the old allocation minimized the priority-weighted error under the old priorities and the new allocation uses the same capacity:

$$\sum_j \mu_j v_j e'_j \;\geq\; \sum_j \mu_j v_j e_j.$$

Or

$$\sum_j \mu_j v_j \left(e'_j - e_j\right) \;\geq\; 0.$$

Splitting the sum into types in S and not in S, we obtain that the error removed from types in S cannot exceed the errors added to those not in S, measured with the old priorities. Otherwise, the priority-weighted error under the old priorities would fall, which contradicts the initial error minimization condition under the old priorities.

The two results in Proposition 2 generate our main result on how errors move when the model is at its capacity. For models with spare capacity, which corresponds to the case in which the priority-weighted error is above its minimum level, the developer can lower the error on certain types without raising errors elsewhere. In contrast, for a model with no excess capacity, a reduction in error from its new priorities must raise priority-weighted error elsewhere, measured with the old priorities, by the same amount or more. Part (i) states where the cost lands, and part (ii) shows that at the capacity frontier, no change in

[5] Since $\ln\lambda' = \ln\lambda + m\ln(1+\beta)$, we have $\lambda' = e^{\ln\lambda + m\ln(1+\beta)} = \lambda\, e^{m\ln(1+\beta)} = \lambda(1+\beta)^m$.

priorities at fixed capacity comes without a cost to other types.

Panel B of Table 1 reports the resulting error rates following two error reports, one from the focal firm and one from an unrelated firm, and the subsequent model updates in our simplified case. With $\beta = 3$ in the simplified case, the developer puts four times the priority $(1+\beta)$ on the reported type. With the hard type making up 25 percent of the inputs of served types, the focal firm's report raises the price of capacity from $1/4 = 0.25$ to $4^{1/4}/4 \approx 0.354$. The error rates of the benchmark and routine types increase by $4^{1/4} \approx 1.41$ times, and the hard type's error rate is multiplied by $4^{1/4}/4 \approx 0.35$. Measured with the old priorities v_j, the report reduces the priority-weighted error on the hard type by 0.032, but adds 0.062 to the benchmark and routine types. The model's error rate in this example, on the other hand, actually decreases from 28.8 to 27.1 percent. This is because the hard type had a high error rate, while benchmark and routine types had low error rates to begin with, so giving more priority to the hard type lowers the model's error rate. On the other hand, if another firm reports errors on the benchmark type, the model's error rate actually increases from 28.8 to 35.6 percent. Thus, the model's error rate can move in either direction depending on the existing priorities and error rates.

4.3 Additional capacity

The results above assume that the capacity is held fixed for every model update. We next ask how additional capacity changes the results.

First, adding capacity alone reduces the error rates of all served types, and may even switch abandoned types to being served. This follows from the fact that the additional capacity lowers the price of capacity λ (the Lagrange multiplier on the capacity constraint), which reduces the error rate λ/v_j of every served type. If λ falls below the priority v_j of an abandoned type, the type becomes served. Thus, the model's error rate falls with capacity, and a firm's map remains correct as long as none of the types it uses newly receives capacity.

Now consider the case in which capacity rises from C to $C + \Delta C$ during the update

following the error report, and the developer updates the model only when capacity has grown enough that no other type's error rate rises.

Corollary 1 (Capacity growth)**.** *Suppose capacity rises from C to $C + \Delta C$, with $\Delta C > 0$, and that no type enters or leaves service.*

(i) Without any change in priority, the error rate of every served type falls by the same factor, and the ranking of the types by error rate remains the same.

(ii) If the developer raises the priority of the reported types in S by the factor $1 + \beta$, with $\beta > 0$ and $m < 1$, the error rate does not rise for any type if and only if the rise in capacity is sufficiently high, $\Delta C \geq \mu(S) \ln(1 + \beta)$.

(iii) Compared with an update that has the same churn but changes neither capacity nor priorities, if the rise in capacity is sufficiently high, the update does not raise any firm's cost (3), whereas at fixed capacity, the same change in priority raises the cost of a firm that uses a served type outside S and no type in S.

With a sufficient capacity increase ΔC, the developer can at least maintain the same capacity allocation for every type in the previous model, even with the additional priority given to the reported types. The threshold corresponds to the capacity that the types in S demand at the old price with their new priorities, which guarantees that capacity need not be taken away from other types. While the error rates do not rise, the firm's map may still become obsolete if the error rate of a reported type falls enough that the ranking changes. However, with a sufficient capacity increase, this does not cost the firm even if its map becomes stale, since no error rate rises: the errors it no longer catches on the type it checks are errors that no longer occur, and its errors on every other type weakly fall, so a stale map only means that the firm gains less than it would after relearning its map. This is in contrast to the fixed capacity case, in which a firm that uses only types outside the report has to pay for the higher error rates.

4.4 Capacity and Delegation

The above results could be interpreted to mean that additional capacity may solve all the issues. However, we show here that what matters is not the absolute level of capacity, but the capacity relative to the complexity of the tasks.

Consider an index of all tasks that a firm could delegate to AI by their difficulty d, measured by the capacity it requires, and let $h = C - d$ be the task's headroom. The headroom is positive when the task requires less capacity than the model has. Now define reliability $q(h)$ as the probability that a model correctly executes a given task with headroom h. It is continuous and rises strictly from zero to one as h rises from $-\infty$ to ∞. A correctly executed task generates a rent $r_C(h) \in (0, \bar{r}]$ for the firm, which is continuous, weakly decreasing in h, and falls to zero as $h \to \infty$ at the same rate at every capacity, so that $\sup_{C \geq C_0} r_C(h) \to 0$, as the task becomes trivial to execute for any competing firm. The firm pays c per error and a fixed cost $g_0 \in (0, \bar{r})$ per delegated task, so that its profit from a task with headroom h is

$$\Pi_C(h) = q(h)\, r_C(h) - c\left[1 - q(h)\right] - g_0. \tag{7}$$

We assume that $\Pi_C(h) > 0$ for some h at every capacity $C \geq C_0$.

Lemma 1 (Bounded headroom)**.** *For any $C \geq C_0$, the headroom of the most profitable task to delegate lies in the same range $[\underline{h}, \bar{h}]$. The difficulty of that task thus stays between $C - \bar{h}$ and $C - \underline{h}$, so it stays within a fixed distance of capacity and grows without bound as capacity grows.*

For tasks that are difficult, the reliability $q(h)$ is low, and it is not profitable to delegate them. For tasks that are trivial, the rent $r_C(h)$ is too low to make it worthwhile. Therefore, if capacity increases, what was once profitable may become trivial enough that it does not generate enough rent. On the other hand, what was previously more difficult may become worthwhile for the firm to delegate. Therefore, there will always be tasks in the firm's consideration, regardless of what the capacity currently is. This lemma is related to the

existing finding that automation leaves people the tasks that cannot be fully automated (Bainbridge, 1983), and is consistent with the models of knowledge hierarchies, in which each level of hierarchy solves problems that the levels below cannot solve (Ide and Talamàs, 2025). Thus, the paper's result on fixed capacity, or the "frontier of capacity," is not a statement about the absolute capacity, but about how difficult the tasks are relative to the capacity of the current model. Therefore, our results that model updates and changing priorities may have negative consequences hold for any model, on the tasks that are most worth delegating.

5 Consequence of Model Update for Firms

In this section, we examine the consequence of model update at capacity on firms that use the model. Given the (assumed) constraints that firms cannot check every input, it catches only the errors that it has learned to look for, and thus the effect of model update on firms depends on how it impacts firm's list and mapping of the errors by input type. For readability, we focus on firms that use at most two types (or tasks), so the ranking of the two types is simple (high or low). We also focus on the case covered in Proposition 2 in which the developer's update follows the input types contained in the error report in S at fixed capacity, and no served type (those that already receive developer capacity for training) is abandoned. We also assume that each firm's map is correct before the update.

We measure the quality of firm's map as the probability that it ranks a failing input above a succeeding one, among the inputs that it has not yet seen. A tie is treated as one half. The quality thus corresponds to the AUC of the map, and equals 100 percent for a perfect map, and 50 percent for a map that performs no better than a random pick. We refer to a re-learned map as the one that correctly reflects the new error rates after an update.

For a firm that uses only the served types (that receive capacity allocation from the developer), Proposition 1 gives us the error rate on its inputs as $\sum_j w_{ij}\lambda/v_j = \lambda\rho_i$, where

w_{ij} is the firm's share of usage in type j (Section 3.2) and

$$\rho_i = \sum_{j=1}^{J} w_{ij} \frac{1}{v_j} \tag{8}$$

is the average of the inverse of the priority across the firm's inputs. Firms with highly firm-specific tasks that are not widely used by other firms may have a high ρ_i, while firms using tasks that are widely used and receive a lot of attention from the developer may have a low ρ_i.

Proposition 3 (Consequences of a model update for firms)**.** *Suppose the developer updates the model based on a report on S, and the update has churn χ.*

(i) (Lists.) The reporting firm's listed failures are corrected. Any other listed failure of type j still fails with probability

$$(1-\chi)\min\left\{1, \frac{e'_j}{e_j}\right\} + \chi\, e'_j,$$

which is below one whenever $\chi > 0$ and $e'_j < 1$, while, if $\chi > 0$ and $e'_j > 0$, some inputs that the firm has seen succeeding now fail.

(ii) (Firms outside the report.) A firm whose types are all served and lie outside S keeps a correct map, and the error rate on its inputs rises by $\lambda\rho_i\big[(1+\beta)^m - 1\big]$, which is proportional to ρ_i.

(iii) (Firms inside the report.) Consider a firm with one served type j in S and another served type l not in S, with $e_j > e_l$, whether it made the report or a competitor did. The firm's map goes stale if and only if $v_l < (1+\beta)v_j$. Its old map then has a lower reliability (AUC) than a relearned map, and the firm's cost with the old map exceeds its cost with a relearned map by $(c - c_h)\, s\, (e'_l - e'_j)$.

Part (i) relates to the firm's list of failed inputs. The developer fixes the failures from the

report, but not the failures outside the report. Each of those is either kept, with probability $1-\chi$, or redrawn (churned), with probability χ. A kept failure still fails if its type's error rate did not fall, while a redrawn (churned) one fails at the new error rate. In the simplified case with $\chi = 0.3$, 30 percent are redrawn, and therefore, some inputs on the firm's list will become obsolete.

Part (ii) then considers a firm whose types are all outside the report. A report on the benchmark type doubles the error rates of the routine and the hard types, so the focal firm's error rate rises from 18.75 to 37.5 percent. Both rates rise by the same factor, so the hard type remains the more error-prone of the two, and the focal firm's map stays correct. The rise is larger for firms whose inputs the developer weighs less, in proportion to ρ_i.

Now suppose the report is on the hard type, by the focal firm or by the competing firm. The report increases the priority of the hard type by four times $(1+\beta)$, from 1 to 4, which is now above the routine type's priority, 2. The error rate of a served type is λ/v, so the type with the higher priority now has the lower error rate. The hard type's error rate falls from 25 to 8.8 percent and the routine type's rises from 12.5 to 17.7 percent, so the focal firm's map, which still ranks hard contracts first, is now wrong. In general, the map goes stale exactly when the report raises the priority of the type the firm checks above the priority of its other type, $(1+\beta)v_j > v_l$. The new error rates do not depend on who made the report, so a competitor's report has the same effect.

The quality of a map is the AUC, which is the probability that it ranks a failing contract above a succeeding one. The map cannot tell two contracts of the same type apart, so only pairs of a hard and a routine contract matter. The focal firm's old map is correct when a hard contract fails and a routine one succeeds, and the new relearned map is correct in the reverse case. After the report, the reverse case is more common, so the AUC of the old map falls from 60 to 40 percent, below the 50 percent of a guess, while a relearned map has an AUC of 60 percent.

The stale map also raises the focal firm's cost. Beyond its list, the focal firm checks hard

contracts, which now fail 8.8 percent of the time, instead of routine contracts, which fail 17.7 percent of the time. It catches fewer errors, and its cost exceeds the cost with a relearned map by $(c - c_h)\, s$ times the 8.8-point gap.

The results separate two kinds of firm knowledge. The list containing specific failures loses value at every update with churn. The map, on the other hand, loses value only when the update targets one type and not the other, whether or not the firm is the one that reports. Thus, an update of the model on completely unrelated areas leaves the firm's map intact, although at capacity it still raises the firm's error rate. On the other hand, a model update that affects the firm's delegated task may hurt the firm, even if the update improves on certain tasks that the firm employs. Laboratory evidence supports this prediction. Bansal et al. (2019) find in an experiment that people learn where AI fails, and an update that makes the AI more accurate overall but introduces new errors harms their performance until they relearn where it fails.

6 Welfare Implications

This section formalizes the welfare implications of the model development. We consider three different groups: firms that use the reported type (focal firm and competing firm), unrelated firms whose tasks all lie outside the report, and users of the model as a whole. If capacity rises with model development in sufficient amount, the cost of errors does not go up (Corollary 1), so we focus on the case with fixed capacity.

Let $\sigma_j > 0$ be the error cost on input of type j for the firms that use it, and $\sigma_j e_j$ is the error cost per input of type j, and $L = \sum_j \mu_j \sigma_j e_j$ is the total cost of errors. The developer's objective from Eq. (2) is the total cost L with σ_j equal to v_j, their priority. For a single firm, their total cost of error is from Eq. (3), with $\theta = (c - c_h)/c$, the share of the error cost the firm saves by catching the error. Let N be the served types that are not in S.

Proposition 4 (Welfare)**.** *Suppose the conditions from Proposition 2 hold. Consider for*

each firm the cost of errors on the inputs it has not yet seen. Let n_{ij} be the share of its inputs that are type j and that it has not seen, with $s \leq n_{ij}$ for the type j it checks, as in Section 3.2.

(i) (Unrelated firms.) For unrelated firms whose types are all served and do not lie in report S, the model update multiplies the cost by $(1+\beta)^m$, so the cost rises by $(1+\beta)^m - 1$ times what they paid before.

(ii) (Firms inside the report.) For firms with input types included in the report as in Proposition 3(iii), the model update lowers the cost with its old map if and only if

$$(n_{ij} - \theta s)\,(e_j - e'_j) \;>\; n_{il}\,(e'_l - e_l).$$

(iii) (All firms.) Let $\bar{\ell}_S$ and $\bar{\ell}_N$ be the weighted (by shares) average cost of errors per input over S and N respectively. The model update lowers the total cost L if and only if

$$\bar{\ell}_S \;>\; R(\beta)\,\bar{\ell}_N, \qquad \textit{where} \qquad R(\beta) = \frac{1-m}{m} \cdot \frac{(1+\beta)^m - 1}{1-(1+\beta)^{m-1}}.$$

The factor $R(\beta)$ exceeds one, rises with β, and tends to one as $\beta \to 0$.

(iv) (Priorities proportional to costs.) If $\sigma_j = \gamma v_j$ for every served type, for some $\gamma > 0$, the update raises L.

Part (i) states the results of Proposition 3(ii) in terms of the cost. For unrelated firms, they experience a rise in error rates across its input types, but the map remains intact. Thus, they pay in terms of higher error rates, but not from a stale map. Part (ii) formalizes the conditions under which even the reporting firm or their competing firms lose even though the update fixes errors on their type. The left-hand side $(n_{ij} - \theta s)(e_j - e'_j)$ is what the firm saves on type j: the fall in the error rate, counted only on the inputs whose errors it does not already catch. The right-hand side $n_{il}(e'_l - e_l)$ is the cost of the new errors on type l, which

reach the firm unchecked. Thus whether they benefit from reporting or receiving developer priorities depends on the trade-off between the two.

Part (iii) shows the conditions in which the total error cost improves for all users. The total cost goes down after an update only if the update moves the capacity towards the types with a higher cost of errors, which happens if the average cost of reported errors sufficiently exceeds those of unreported errors, and how much it must exceed depends on how much priority the reported errors receive. Finally, Part (iv) points out that if the developer priority already aligns proportionally with the error costs, then changing the priority will lead to an increase in costs.

Overall, these results show that how the model updates change welfare of users and firms depends critically on the developer priorities.

7 Evidence from Large Language Models

In this section, we test the theoretical predictions with large language models. The LLMs in production differ from our theory in various ways. First, in our theory, the developer can choose how much capacity to allocate to each type, and the error rates are determined by the allocation. In the real world, developers do not choose exact capacity or parameters by task type. Instead, developers set how much weight each kind of input receives in the training process, and training sets the LLM's parameters to fit the training data.[6] Thus, in our empirical test, we set the developer priorities only through the training weights, and observe the outcome of the trained LLM. Second, the inputs that firms send to the model are not labeled by type. Within a task, nothing marks an input as hard or routine, because what makes an input hard is whether the model gets it wrong. Thus, in our test, we do not label the inputs within a task by type, and each firm learns from the inputs it has experienced

[6]There are some exceptions. Task-specific adapters allocate some parameters to a task (Houlsby et al., 2019; Hu et al., 2022), but the major general-purpose models answer the many different tasks that firms and people use them for with the same parameters. Mixture-of-experts models send different inputs to different subnetworks (Shazeer et al., 2017; Fedus et al., 2022), though how the inputs are assigned to subnetworks is still learned as part of the training process.

which kinds of input fail. Third, our theory describes a single update. In our empirical test, we examine how the error allocation evolves over multiple rounds, in which the firm reports its current failures in each round, so the inputs (and the types of inputs) that get reported can change from round to round.

7.1 Design

We fine-tune various open-source large language models as a means of introducing model updates. We use Pythia models with 70, 160, and 410 million parameters, which are pre-trained on the same public texts (Biderman et al., 2023). We also use Qwen3 with 0.6 billion parameters (Yang et al., 2025), a recent model released for real world use as additional evidence. We use eight text classification tasks as our input types: identifying sentiment in movie reviews (Socher et al., 2013), judging grammatical acceptability (Warstadt et al., 2019), categorizing news articles into different topics (Zhang et al., 2015), identifying emotion in tweets (Saravia et al., 2018), detecting irony, offensive language, and sentiment in tweets (Barbieri et al., 2020), and telling opinion sentences in movie reviews apart from plot descriptions (Pang and Lee, 2004).

For each seeded run, we draw 1,000 training examples per task that the developer uses for training the model, and 2,000 deployment examples per task that the firms actually use in their work. All tasks are put through the same LLM, and the resulting output is put through each task's own output layer that converts it into the answers (e.g., movie-review sentiment being positive vs. negative, or news topic into four different categories). The developer fine-tunes every model with the same budget. The Pythia models at different sizes give us a way to vary model capacity. For Qwen3, we generate models with different capacity by training a version of Qwen3 in which only the top layer (among 28 transformer layers) can be fine-tuned. The Pythia models also differ in their pretraining learning rate, and the restricted version of Qwen3 has a smaller optimization budget, so neither comparison varies capacity alone.

Four tasks are randomly chosen as benchmarked tasks, with priority equal to 3 in training, and the other four have priority of one. The focal firm uses two benchmarked tasks, and the unrelated firm uses two tasks that are not benchmarked. The Pythia models perform poorly on the grammatical acceptability task (close to always flagging a sentence as acceptable), so we treat it as an abandoned task, keeping it in training but never assigning it to a firm. Each firm has experienced half of its inputs, and its list contains the inputs with errors among that half. A report then contains these errors, which the developer adds to its training data with four times the priority, and the developer continues training the existing model, which corresponds to a patch.

A firm's map ranks future inputs by how likely the model is to fail on them, based on its experience. Given that we do not label the inputs within a task by type in our empirical test, we need a way to come up with the map from the input texts directly. To do so, we generate each firm's map as a logistic regression on the sentence embeddings of the texts of inputs that the firm has experienced (Reimers and Gurevych, 2019; Wang et al., 2020), to determine whether the model is likely to be wrong or correct. We then measure the AUC on the remaining examples the firm has not experienced. Initially, before any update, the firm's map has an AUC of about 60%, and thus it is a weak but useful predictor of future errors.

We then compare the effects following the focal firm's report versus no report, and also the report from a competing firm that uses the same tasks. We also compare the focal firm's reporting through six rounds versus six rounds without any error report, as well as six rounds of reports from a third firm that uses tasks that neither firm uses. To separate the effect of the added examples from that of their higher priority, in the two smaller Pythia models we also compare runs that add the same reported examples at their normal priority. We run 20 seeds for the two smaller Pythia models, and 10 for the largest Pythia model with 410 million parameters and for each version of Qwen3.

7.2 Priorities and Initial Error Rates

Table 2 reports the allocation slope and the results from a single error report. For each seed we randomly assign weights evenly spaced on a log scale from 0.35 to 2.83, such that the highest weight is eight times the lowest. This generates variations in the priorities that we use to estimate the sensitivity of the task error to different levels of priorities (the allocation slope), which is calculated as the regression of the log of the task's error rate on the log of its priority with task and seed fixed effects.[7] We confirm that the error rate falls with priority, with the slopes at -0.14, -0.18, and -0.12 in the three Pythia models with 70, 160, and 410 million parameters, and -0.11 and -0.15 for the Qwen3 models with full layers and one layer, respectively. The error rates before any report range from 20% for the full Qwen3 model to 37% for the smallest Pythia model.

7.3 Error rates following update

We then examine how the errors change following an error report. First, we examine what happens to the error rates on the ones that are reported as errors. For the reported errors, the error rates not surprisingly fall across all models, and the error reduction is greater for the largest models (Pythia 410m and Qwen3 full). Furthermore, the error rate reduction is greater following firm's own error report than after an update without a report. For example, 79.8% of inputs that failed before still fail following an update without a report, but only 56.5% following firm's own reporting. The error rate falls in general, regardless of whether the firm itself reported since additional training may increase model performance, as well as the overall churn.

On the other hand, the error rates on the inputs that the firm has previously seen succeeding increase across all models, again partly due to churn, and partly due to not receiving more weights in training. For example, for Pythia 70m, the error rate on the

[7] The regression excludes the grammatical acceptability task, which is an abandoned task.

Table 2: Model size and the effects of one report

	Pythia, parameters			Qwen3, 0.6 billion	
	70 million	160 million	410 million	Full	One layer
Seeds	20	20	10	10	10
Error before any report, all tasks (%)	37	31	22	20	31
Allocation slope	−0.14	−0.18	−0.12	−0.11	−0.15
	(0.01)	(0.02)	(0.03)	(0.02)	(0.01)
One report (%)					
Error rate on the firm's seen failures					
After an update without a report	79.8	75.6	82.2	83.3	92.1
After the firm's own report	56.5	41.1	16.3	12.4	69.5
Error rate on the firm's seen successes					
After an update without a report	10.0	7.7	4.5	6.5	3.5
After the firm's own report	28.9	26.1	11.8	16.0	19.5
Error rate on the firm's unseen examples					
Before the update	38.7	30.2	20.3	26.9	41.6
After an update without a report	37.0	28.0	20.3	27.0	40.5
After the firm's own report	41.8	32.6	21.6	28.7	42.7
AUC of the firm's map on its unseen examples					
Before the update	63.3	63.2	60.8	59.2	60.2
After an update without a report	61.7	61.8	60.8	59.0	59.7
After the firm's own report	53.9	56.5	60.4	58.1	56.2
Own report minus an update without a report (percentage points)					
Loss of AUC of the firm's map	7.8	5.3	0.4	1.0	3.5
	(0.8)	(0.8)	(0.5)	(0.4)	(0.4)

This table reports the results of the experiments on large language models. The LLMs used are Pythia models with 70, 160, and 410 million parameters, and Qwen3 with 0.6 billion parameters, one with full layers and a limited one with only the last layer being fine-tuned. We report the seeds (runs), the allocation slope, and the model errors before any report across the eight text classification tasks we use. We then report the various error rates and map reliability (AUC) after an error report across different inputs: those that firms have already seen versus those that they have not. The rows compare a firm's own report with an update without a report, pooling the focal and the unrelated firms. See Section 7.1 for details. Means across seeds, with standard errors in parentheses. Where the focal and unrelated firms are pooled, and for the allocation slope, the standard errors are clustered by seed. Error before any report is the deployment error of the round-zero model over all eight tasks. The allocation slope is the coefficient in a regression of the log of a task's deployment error on the log of its priority, its weight in training, with task and seed fixed effects, when the weights are spread over a ratio of eight and assigned to tasks at random. The regression excludes grammatical acceptability, which the Pythia models abandon. The single-report rows compare a firm's own report with an update with no report, pooling the focal and unrelated firms. Seen examples are the half of the firm's examples that it has experienced, and unseen examples are the other half. Seen failures are the seen examples that the model got wrong before the update, which are the examples in the firm's own report, and seen successes are those it got right, so their error rates before the update are 100 and 0 percent. The AUC rows give the AUC of the map that the firm fitted before the update, as a predictor of the failures of the model before the update and after it, and the loss of AUC of the firm's map is the fall in that AUC after the firm's own report minus the fall after an update without a report. The Qwen3 columns draw both firms from the four tasks, other than grammatical acceptability, on which its error is highest, so their firm-level rows are not directly comparable with the Pythia columns. In the Full column the developer fine-tunes the whole model. In the One layer column it trains only the top one of the model's 28 transformer layers, together with the final normalization and the task classifiers.

previously successful inputs increases to 10% (from zero) following an update without a report, but increases to 28.9% following its own report.

In addition, the error rates on the firm's unseen examples also rise across all models. For example, before the update, Pythia 70m reports 38.7% error rate on the ones unseen, which rises to 41.8% following firm's own error report. For the full Qwen3 model, it rises from 26.9% to 28.7%. This is consistent with Proposition 2 that the model update reallocates errors from the reported inputs to the inputs outside the report, including the firm's own.

Similarly, in looking at the map quality, the AUC of firm's existing map falls across all models, from 63.3% to 53.9% for Pythia 70m, a fall of 7.8 percentage points more than after an update without a report. The fall in the map quality gets smaller with larger model capacity. For Pythia 410m, the loss of AUC is only 0.4 percentage points, and for the full Qwen3 model, the loss is 1.0 percentage point. The results are also similar if the report comes from a competitor rather than from firm's own, as long as the competitor shares the same input types (Internet Appendix).

Overall, these results highlight the effect of model update on the relocation of errors, from the ones that are reported to the ones that are not reported. This relocation of error and the adverse effect on firm's own mapping are greatest in the models with the least capacity.

7.4 Repeated updates

We now examine the results following six rounds of model updates. Figure 1 plots how the errors and the map reliability (AUC) of the focal firm change over the six rounds of model updates using Pythia models relative to the six rounds update with no reports. Table 3 reports the changes following six rounds.

As seen in the first row of Figure 1, across all three models, the focal firm's errors on the inputs it has not seen initially increase after the first round relative to the no reporting case, just as our theory predicts and we see in the previous subsection. However, the error

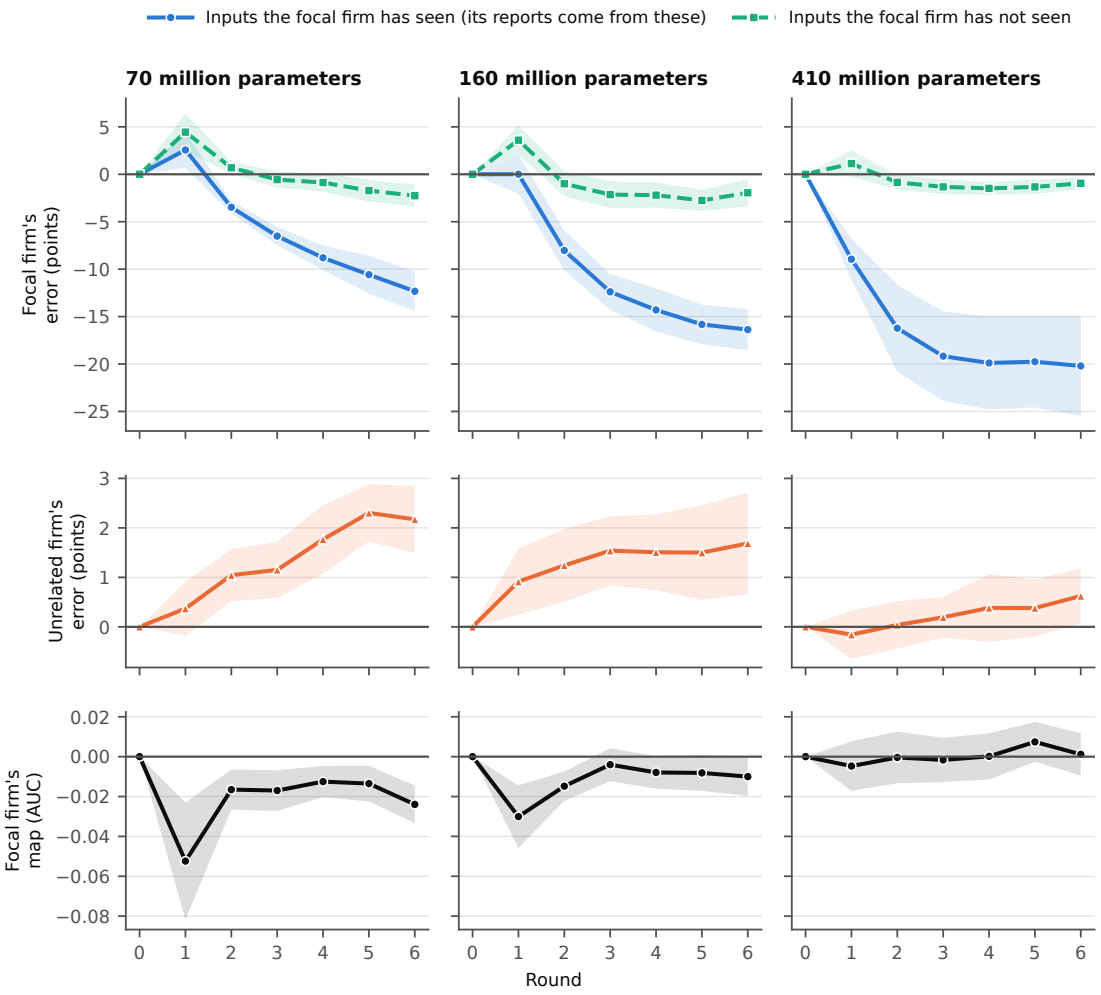


Figure 1: Repeated patching in language models of three sizes

This figure reports the changes in error rates and map reliability across the six rounds of model updates, relative to six rounds of updates without reports. The first, second, and third columns use Pythia models with 70, 160, and 410 million parameters, respectively. The first row reports the focal firm's error on the inputs it has seen and on those it has not seen. The second row reports the unrelated firm's error rates. The last row reports the focal firm's original map reliability (AUC) over the six rounds. See Section 7.4 for details on our experiments. The focal firm reports the failures it has learned in each round, and every example it has reported keeps four times its task's weight in every later round. Every panel shows the reporting chain minus the chain without reports, so the zero line is continued training without reports. Changes in error are in percentage points since round zero. The bands are 95 percent confidence intervals across 20, 20, and 10 seeds. Without reports, the AUC of the original map is about 0.6, and a coin flip scores 0.5.

rates subsequently decrease with each round of update. Since we are assuming that the set of inputs remain fixed across multiple rounds, firm can report the ones that broke from the first round, and repeat the process after multiple rounds, so that it can correct the relocated errors over time.

On the other hand, the changes in the map quality after six round remain negative in the two smaller models. Because the reports fix the input types that the firm's map ranks the highest (e.g., the hard task), the remaining failures occur more in the safer type (routine task). Thus, the firm's errors fall through the rounds, but their existing map is pointing to the wrong place to look, which costs the firm. For the largest model, however, this does not occur, and the change in AUC is close to zero. Given the spare capacity of the largest model, the model updates fix the reported errors without affecting other inputs too much, and the map remains correct.

While the error rates fall for the focal firm after six rounds, the error rates for an unrelated firm that did not report increase over the rounds, much more so for the smaller models. The logic is again the same: fixing the reporting firm's errors takes capacity away from other unreported tasks, and therefore reduces the performance for other firms that did not receive developer priority. On the other hand, given that the error rates for the unrelated firm rise in about the same proportion across all types, the mapping does not deteriorate (Table 3, Panel A).

The reported errors change two things for the developer. First, it adds more data to train on. Second, it changes their priority. To examine whether the changes that we observe are driven by the pure addition of more training data, we examine the case in which each update adds the same reported inputs to the training data while keeping the priority fixed. We then net out the changes in this matched chain to tease out the effect of changing priorities from the total effect. In the two smaller Pythia models, the change in priority alone raises the unrelated firm's error by 1.7 and 1.5 percentage points, most of the total rise, while the

Table 3: Six rounds of reporting, relative to training without reports and to a matched chain

	Pythia, parameters			Qwen3, 0.6 billion	
	70 million	160 million	410 million	Full	One layer
Panel A: Six reports by the focal firm, minus the no-report chain					
Focal firm					
Change in error, examples it has seen	−12.3	−16.4	−20.2	−24.9	−20.0
	(1.0)	(1.0)	(2.3)	(1.1)	(0.5)
Change in error, examples it has not seen	−2.3	−1.9	−1.0	−1.1	−1.2
	(0.6)	(0.7)	(0.3)	(0.4)	(0.5)
Change in AUC of its original map	−2.3	−1.3	0.0	−0.6	−1.2
	(0.4)	(0.3)	(0.3)	(0.3)	(0.2)
Unrelated firm					
Change in error	2.2	1.7	0.6	0.4	1.1
	(0.3)	(0.5)	(0.2)	(0.2)	(0.2)
Change in AUC of its previous round's map	0.7	0.9	−0.1	−0.5	0.7
	(0.3)	(0.2)	(0.4)	(0.3)	(0.1)
Panel B: Six reports by the focal firm, minus the matched chain					
Focal firm					
Change in error, examples it has seen	−2.8	−2.6	n.a.	n.a.	n.a.
	(0.5)	(1.1)			
Change in error, examples it has not seen	0.2	−0.4	n.a.	n.a.	n.a.
	(0.4)	(0.7)			
Change in AUC of its original map	−1.1	−0.5	n.a.	n.a.	n.a.
	(0.3)	(0.2)			
Unrelated firm					
Change in error	1.7	1.5	n.a.	n.a.	n.a.
	(0.3)	(0.5)			

This table reports the results from running six rounds of reporting and model updates. Panel A reports the changes in error rates and map reliability (AUC) for the focal firm and the unrelated firm after the focal firm reports errors for six rounds versus the runs without any report. Panel B reports the same results but relative to the matched chain in which the reported errors are used to train but they do not receive additional priority, for the two smaller Pythia models. See Section 7.4 for details. Means across seeds in percentage points, with standard errors in parentheses. Panel A gives six rounds of reporting by the focal firm minus six rounds of the same training without any report (the no-report chain). Panel B gives the same reporting chain minus a matched chain, which adds in every round the same reported examples at their task's usual weight, with the same starting model, seeds, and training budget, so its rows measure the effect of the change in priority alone. Panel B covers only the two smaller Pythia models, and n.a. marks the others. The focal firm reports in every round, and every example it has reported keeps four times its task's weight in every later round. Changes in error run from round zero to round six. The original map is the focal firm's map fitted before any report, and the change in its AUC is averaged over rounds one to six. The unrelated firm's previous round's map is the map it fitted on the model of the round before, and the change in its AUC is averaged over rounds one to six. The Internet Appendix gives the effects of a third party's reports and repeats Panel A with weights that compound. The columns are as in Table 2.

added data alone raise it by only 0.4 and 0.2 points (Table 3, Panel B).

These results show that the unrelated firms and focal firms pay the price in different ways. Reporting firms benefit from error reduction over repeated rounds, but in the models that fit their training data poorly, they pay the cost when their existing knowledge of where the model fails (map) becomes obsolete. For unrelated firms, their existing knowledge of where the model generally fails remains intact, but they pay the price of higher error rates across all of their tasks.

7.5 Reporting firm's unreported tasks

The results so far allow the focal firm to report failures from all of its tasks. We next examine what happens if it can only report on one task. This can arise when a firm only checks one type of input and not others due to the cost of checking. For example, a firm may see that its routine contract reviews work well, so does not bother checking their quality thoroughly, and instead checks only the hard contracts. To do so, we force the focal firm to only report from one of its two tasks, chosen at random.

We report the results in the Internet Appendix. For the two smaller Pythia models, we find that the model update increases the error on the other (unreported) task by 0.6 percentage points after the first round, and it remains higher after six rounds in the 70 million parameter model. However, the unreported task behaves like the tasks of the unrelated firm: while its error rate increases, its map remains intact. On the other hand, for the reported task, the map loses 2.5 and 1.8 percentage points of AUC in the Pythia 70m and 160m models.

7.6 Costs to firms

We finally translate the above results into actual costs for firms following Proposition 4. We count each error that is not caught at the cost of $c = 1$ and the ones it catches at $1 - \theta$ with $\theta = 0.8$, such that catching an error saves the firm 80% of the cost. All the errors on the list

are caught. Beyond the list, the firm checks 30% of the inputs unseen in each task that the firm's map ranks as being most likely to fail.[8]

Table 4 reports the differences in the error rates and the cost after its own report relative to an update without a report. Panel A examines the costs for the reporting firm. For all models, the error rate drops for the inputs that are checked (ones with highest risk), but increases for the inputs that are not checked. This translates into higher cost for the firm if they keep using the existing list and map, and is greater for models with more binding capacity. For example, the increase in the cost for the smallest Pythia model is 7.3 percentage points versus 1.9 points for the largest Pythia model, and 5.0 versus 2.3 points for the limited vs. full Qwen3 models.

Table 4 Panel B reports the cost for the other firms. For unrelated firms, the cost goes up slightly after the update, but only in the smaller Pythia models. For the largest models (Pythia 410m and full Qwen3) the cost does not rise, and in full Qwen3 it actually goes down. The increase in cost is the greatest for the competing firms across all models but is greater once again for the models at capacity. For the smaller Pythia models the increases in cost for competing firms are 7.8 and 8.2 percentage points respectively, and 3.2 points for the largest model. The increases in costs again are greater when they rely on their existing list and map, and are reduced if they relearn their list and map. Lastly, we also examine the changes to the error rates for all users. We find that the total error rate goes up for the models at capacity (two smaller Pythia models and limited Qwen3), but decreases for models with spare capacity. If the error costs on all tasks were equal, this would imply that the total error cost for all users goes up for the models at capacity and goes down for the others, but in general the change in the aggregate cost for all users depends on the actual cost structure (Proposition 4(iii)). Overall, these results highlight the effects of LLM development and developer priorities on firms that incorporate and delegate their work to AI models.

[8]The results are similar when the firm checks none, 10, 20, or 50 percent of its unseen inputs beyond its list.

Table 4: Welfare Implications of model update

	Pythia, parameters			Qwen3, 0.6 billion	
	70 million	160 million	410 million	Full	One layer
Δ: own report minus an update without a report (percentage points)					
Panel A: The reporting firm					
Cost of errors after an update without a report	19.0	14.5	10.4	14.1	19.2
	(0.5)	(0.6)	(0.6)	(0.2)	(0.2)
Δ Error rate, all inputs	3.5	3.1	−3.3	−5.3	1.1
	(0.5)	(1.0)	(0.4)	(0.5)	(0.4)
On inputs it checks	−4.8	−5.2	−6.6	−9.5	−4.9
	(0.2)	(0.3)	(0.4)	(0.1)	(0.2)
On inputs it does not check	8.3	8.4	3.3	4.2	6.0
	(0.6)	(1.1)	(0.3)	(0.5)	(0.5)
Δ Cost of errors, with its old list and map	7.3	7.3	1.9	2.3	5.0
	(0.6)	(1.0)	(0.3)	(0.5)	(0.5)
Δ Cost of errors, with a relearned list and map	2.1	1.9	−0.2	−0.5	1.2
	(0.3)	(0.4)	(0.2)	(0.2)	(0.2)
Cost of stale knowledge	5.2	5.4	2.2	2.8	3.8
	(0.4)	(0.6)	(0.2)	(0.3)	(0.3)
Panel B: Other firms					
Δ Cost of errors, unrelated firm	0.3	0.4	−0.1	−0.7	0.1
	(0.2)	(0.2)	(0.1)	(0.1)	(0.1)
Δ Cost of errors, competing firm, with its old list and map	7.8	8.2	3.2	3.4	5.5
	(0.9)	(0.8)	(0.4)	(0.2)	(0.5)
Δ Cost of errors, competing firm, with a relearned list and map	2.4	2.6	0.8	0.6	1.6
	(0.4)	(0.3)	(0.2)	(0.1)	(0.2)
Δ Error rate, all tasks	1.4	1.3	−1.0	−1.8	0.5
	(0.3)	(0.4)	(0.2)	(0.2)	(0.1)

This table reports the effect of model update on firm costs. Panel A examines the differences in the error rates and on costs for reporting firms for a model update after a report versus a model update without their report. Panel B reports the difference in the costs of errors for unrelated firms who do not share any input with reporting firms and for competing firms who use the same tasks as the reporting firm. See Section 7.6 for details. Means across seeds in percentage points of the firm's inputs, and in the last row of all inputs outside grammatical acceptability, with standard errors clustered by seed in parentheses. The focal and the unrelated firm each report in a separate run, and the two are pooled, as in Table 2. An error that the firm does not catch costs one, and an error that it catches costs $1 - \theta$, with $\theta = 0.8$. The firm catches the errors on its list, which are its seen failures, and on the 30 percent of its unseen inputs in each task that its map ranks highest; these are the inputs it checks. The two rows below the error rate split its change between the inputs the firm checks and those it does not, each as a share of all its inputs, using the old map, which is the map fitted before the update. With a relearned list and map, the firm observes its seen inputs again after the update: its list is the seen inputs that now fail, and its map is refitted on them, in the same way after the update without a report. The cost of stale knowledge is the difference between the two rows of the change in cost; it does not count the cost of observing the seen inputs again. The unrelated firm uses its old list and map. For the competing firm, a same-task competitor reports the same number of its own failures in the firm's two tasks, and the firm's lists and maps are as in Panel A. All tasks exclude grammatical acceptability. The columns are as in Table 2.

8 Implications and discussions

If the developer's new priorities align with the costs that the errors impose on its users, then its priority-driven model updates would be efficient. If the old priorities already aligned, then changing the priority during a model update at capacity will raise the total cost of errors (Proposition 4). However, our results point to three reasons why the relocation of error can be costly. First, an update can make a firm's own knowledge of where AI fails obsolete, which may not be fully anticipated by the firm. Second, given that one model serves many firms, a single priority is unlikely to match the cost structure of all the firms across the board, so a relocation of error, even if it is efficient on average, is going to generate winners and losers, which is important to understand. Third, developer priorities are likely to change in favor of certain firms over others, and thus the fix for them may come at the cost of other firms. In this section, we outline some of the practical implications for AI model development and for firms that use these models.

Knowledge about a model has a shelf life. What a firm knows about where the model fails can lose its value at each model update, and the loss is driven by the developer's own priorities. A capacity-driven model update lowers firms' overall error rates while leaving their maps intact (Corollary 1), but a priority-shift-driven model update lowers the value of existing knowledge (Proposition 3). Given that developers of frontier models often do not disclose their priorities for each model update, a firm cannot predict in advance how model updates will change its existing knowledge. Thus, this is an important risk for firms to manage and monitor in incorporating AI into their organizations. After each update, however, a firm can partly tell which kind of update it faces: if its error rates rise or fall in about the same proportion across its types, its map still holds, but if their order changes, it must relearn where the model fails.

Reporting errors has costs. Reporting model failures to the developer may lead to fixes on those reported inputs, but at model capacity, it can break other inputs, including ones that already work well. Thus, a firm's reporting decision should weigh these costs against the fixes. The fixes are worth less to the firm than they may appear since the fixes are done on the errors that the firm already catches, and as we find in our experiments the cost actually goes up across all models if it keeps its old knowledge (Table 4). The loss is mitigated if the firm checks its seen inputs and updates its knowledge, but that may be costly. A developer can potentially reduce the harm to its reporting users by using lighter patches, and by asking not just for their failures but their successes as well. Firms that do not report, or have low priority with the developer, must monitor their model usage carefully after each update, even in tasks that already work well.

Disclosure helps, but does not fix everything. Gans (2026) proposes that developers should publish how well their model performs on each category of task, so that organizations can better decide where to incorporate the models. In our model, such disclosure would help firms to see when the ranking of which type fails most changes, and make necessary checks. However, first, disclosure does not change where the errors would go, so the firms with low priorities still bear more errors than those with high priority. Second, the published categories are likely to be significantly coarser than firms' own input types. Our results show that which inputs fail most can change even within a task, which a task-level disclosure would not show.

Firms with niche uses bear the most. For firms that use AI models in unconventional ways, their tasks are by design likely to receive little priority, and too few reports to raise it. Thus, the relocation of errors is likely to be more costly to firms with more specialized tasks. For these firms, and for other firms in general, seeking out models whose priorities are not dominated by a few large users may be one way to protect against relocation of errors.

9 Conclusion

Large language models are being utilized in many different ways, and are quickly changing the ways in which organizations work. LLMs by design do not optimize their performance for every task that they are asked to perform. Instead, they generate heuristics, or predictions, which are bound to leave errors on certain tasks or inputs they are asked to handle. Finding out where the AI models perform well is therefore increasingly important knowledge for modern firms in the age of AI. This paper examines how the value of this knowledge may deteriorate as models advance and update more often. We do so through a model of how models improve with changing developer priorities. Our main prediction is that at the capacity frontier, improving a model reduces the error rates on some inputs but increases errors on others, especially for firms whose failures the developer does not prioritize. Moreover, even reports by firms themselves and by their competing firms can make the model fail on inputs that used to work, and make firms' knowledge of where it fails obsolete. An update at model capacity lowers the total cost of errors only when the developer priority aligns with how much the errors cost. We test these predictions empirically with two families of large language models, and find patterns consistent with our predictions for the ones in which capacity binds.

Our results have broader implications for progress in AI. Because whether capacity binds depends on the work a model is asked to do and on what each update can change, no model is large enough to be exempt from the costs of model updates that we show. The models that absorbed the error reports in our paper without relocating error did so relative to eight simple tasks. A model in practice that serves a wide range of uses and is updated with limited budget can be small relative to that range of uses. Therefore, models that advance with better training and data may also be reallocating where the errors occur. Where those errors go and who bears the cost is driven by the developer's priorities, which are often undisclosed and change frequently. Progress in AI is therefore a question not only of what newer models can do, but also of where they fail and who bears those costs.

References

Bainbridge, Lisanne, "Ironies of Automation," *Automatica*, 1983, *19* (6), 775–779.

Bansal, Gagan, Besmira Nushi, Ece Kamar, Daniel S. Weld, Walter S. Lasecki, and Eric Horvitz, "Updates in Human-AI Teams: Understanding and Addressing the Performance/Compatibility Tradeoff," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019, *33* (1), 2429–2437.

Barbieri, Francesco, Jose Camacho-Collados, Luis Espinosa Anke, and Leonardo Neves, "TweetEval: Unified Benchmark and Comparative Evaluation for Tweet Classification," in "Findings of the Association for Computational Linguistics: EMNLP 2020," Association for Computational Linguistics, 2020, pp. 1644–1650.

Biderman, Stella, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, Aviya Skowron, Lintang Sutawika, and Oskar van der Wal, "Pythia: A Suite for Analyzing Large Language Models Across Training and Scaling," in "Proceedings of the 40th International Conference on Machine Learning," Vol. 202 of *Proceedings of Machine Learning Research*, 2023, pp. 2397–2430.

Bryan, Kevin A. and Joshua S. Gans, "Training AI for When Humans Will Use It," arXiv:2608.12538v1, 2026. Version of August 14. Earlier version: NBER Working Paper 35490, July 2026.

Chen, Lingjiao, Matei Zaharia, and James Zou, "How Is ChatGPT's Behavior Changing Over Time?," *Harvard Data Science Review*, 2024, *6* (2).

Dell'Acqua, Fabrizio, Edward McFowland III, Ethan Mollick, Hila Lifshitz, Katherine C. Kellogg, Saran Rajendran, Lisa Krayer, François Candelon, and Karim R. Lakhani, "Navigating the Jagged Technological Frontier: Field Experimental Evidence of the Effects of Artificial Intelligence on Knowledge Worker Productivity and Quality," *Organization Science*, 2026, *37* (2), 403–423.

Echterhoff, Jessica, Fartash Faghri, Raviteja Vemulapalli, Ting-Yao Hu, Chun-Liang Li, Oncel Tuzel, and Hadi Pouransari, "MUSCLE: A Model Update Strategy for Compatible LLM Evolution," in "Findings of the Association for Computational Linguistics: EMNLP 2024," Association for Computational Linguistics, 2024, pp. 7320–7332.

Fedus, William, Barret Zoph, and Noam Shazeer, "Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity," *Journal of Machine Learning Research*, 2022, *23* (120), 1–39.

Gabaix, Xavier, "A Sparsity-Based Model of Bounded Rationality," *Quarterly Journal of Economics*, 2014, *129* (4), 1661–1710.

Gans, Joshua S., "Artificial Jagged Intelligence: When AI Benchmarks Misstate Deployment Value," Working Paper 34712, National Bureau of Economic Research, 2026. Revised June 2026.

Gigerenzer, Gerd and Henry Brighton, "Homo Heuristicus: Why Biased Minds Make Better

Inferences," *Topics in Cognitive Science*, 2009, *1* (1), 107–143.

Han, Henry, "LLMs as Bounded Rational Agents: Governing Bias as a System Property," SSRN Working Paper 5541298, 2025.

Houlsby, Neil, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly, "Parameter-Efficient Transfer Learning for NLP," in "Proceedings of the 36th International Conference on Machine Learning," Vol. 97 of *Proceedings of Machine Learning Research*, 2019, pp. 2790–2799.

Hu, Edward J., Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen, "LoRA: Low-Rank Adaptation of Large Language Models," in "International Conference on Learning Representations," 2022.

Ide, Enrique and Eduard Talamàs, "Artificial Intelligence in the Knowledge Economy," *Journal of Political Economy*, 2025, *133* (12), 3762–3800.

Kiela, Douwe, Max Bartolo, Yixin Nie, Divyansh Kaushik, Atticus Geiger, Zhengxuan Wu, Bertie Vidgen, Grusha Prasad, Amanpreet Singh, Pratik Ringshia, Zhiyi Ma, Tristan Thrush, Sebastian Riedel, Zeerak Waseem, Pontus Stenetorp, Robin Jia, Mohit Bansal, Christopher Potts, and Adina Williams, "Dynabench: Rethinking Benchmarking in NLP," in "Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies," Association for Computational Linguistics, 2021, pp. 4110–4124.

OpenAI, "Expanding on What We Missed with Sycophancy," OpenAI blog, May 2025. Accessed 2026-10-22.

_____, "Sycophancy in GPT-4o: What Happened and What We're Doing About It," OpenAI blog, April 2025. Accessed 2026-10-22.

Ott, Simon, Adriano Barbosa-Silva, Kathrin Blagec, Jan Brauner, and Matthias Samwald, "Mapping Global Dynamics of Benchmark Creation and Saturation in Artificial Intelligence," *Nature Communications*, 2022, *13*, 6793.

Pang, Bo and Lillian Lee, "A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts," in "Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)," Association for Computational Linguistics, 2004, pp. 271–278.

Reimers, Nils and Iryna Gurevych, "Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks," in "Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)," Association for Computational Linguistics, 2019, pp. 3982–3992.

Rubinstein, Ariel, "Finite Automata Play the Repeated Prisoner's Dilemma," *Journal of Economic Theory*, 1986, *39* (1), 83–96.

Salant, Yuval, "Procedural Analysis of Choice Rules with Applications to Bounded Rationality," *American Economic Review*, 2011, *101* (2), 724–748.

Saravia, Elvis, Hsien-Chi Toby Liu, Yen-Hao Huang, Junlin Wu, and Yi-Shin Chen, "CARER: Contextualized Affect Representations for Emotion Recognition," in "Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing," Association for Computational Linguistics, 2018, pp. 3687–3697.

Shazeer, Noam, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer," in "International Conference on Learning Representations," 2017.

Simon, Herbert A., "A Behavioral Model of Rational Choice," *Quarterly Journal of Economics*, 1955, *69* (1), 99–118.

Socher, Richard, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts, "Recursive Deep Models for Semantic Compositionality over a Sentiment Treebank," in "Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing," Association for Computational Linguistics, 2013, pp. 1631–1642.

Truong, Sang, Serena Wang, Nick Haber, and Sanmi Koyejo, "Strategic Evaluation: Incentivizing AI Capability Coverage with Private Benchmarks," in "Advances in Neural Information Processing Systems," 2026. Forthcoming.

Tversky, Amos and Daniel Kahneman, "Judgment under Uncertainty: Heuristics and Biases," *Science*, 1974, *185* (4157), 1124–1131.

Wan, Alexander, Kevin Klyman, Sayash Kapoor, Nestor Maslej, Shayne Longpre, Betty Xiong, Percy Liang, and Rishi Bommasani, "The 2025 Foundation Model Transparency Index," arXiv:2512.10169, 2025.

Wang, Jiacheng and Jinbin Huang, "Reward Hacking as Equilibrium under Finite Evaluation," arXiv:2603.28063, 2026.

Wang, Wenhui, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou, "MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pre-Trained Transformers," in "Advances in Neural Information Processing Systems," Vol. 33, Curran Associates, Inc., 2020, pp. 5776–5788.

Warstadt, Alex, Amanpreet Singh, and Samuel R. Bowman, "Neural Network Acceptability Judgments," *Transactions of the Association for Computational Linguistics*, 2019, *7*, 625–641.

Wilder, Bryan, Eric Horvitz, and Ece Kamar, "Learning to Complement Humans," in "Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20)," International Joint Conferences on Artificial Intelligence Organization, 2020, pp. 1526–1533.

Yang, An, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui et al., "Qwen3 Technical Report," arXiv:2505.09388, 2025.

Zhang, Xiang, Junbo Zhao, and Yann LeCun, "Character-Level Convolutional Networks for Text Classification," in "Advances in Neural Information Processing Systems," Vol. 28, Curran Associates, Inc., 2015.

Zhong, Ruiqi, Dhruba Ghosh, Dan Klein, and Jacob Steinhardt, "Are Larger Pretrained Language Models Uniformly Better? Comparing Performance at the Instance Level," in "Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021," Association for Computational Linguistics, 2021, pp. 3813–3827.

Zhuang, Simon and Dylan Hadfield-Menell, "Consequences of Misaligned AI," in "Advances in Neural Information Processing Systems," Vol. 33, Curran Associates, Inc., 2020, pp. 15763–15773.

A Proofs

Throughout, $I = \{j : v_j > \lambda\}$ is the set of served types, $\mu_I = \sum_{j \in I} \mu_j$ is their total share, and $\mu(S) = \sum_{j \in S} \mu_j$, so that $m = \mu(S)/\mu_I$ and $\mu_I m = \mu(S)$.

Proof of Proposition 1. Each term $\mu_j v_j e^{-k_j}$ of the objective in (2) is strictly convex in k_j, because $\mu_j v_j > 0$, so the objective is strictly convex in $(k_1, \ldots, k_J)$. The feasible set is convex and compact. A unique minimizer therefore exists and is characterized by the Kuhn–Tucker conditions. With multiplier $\lambda \geq 0$ on the capacity constraint, these conditions are

$$v_j e^{-k_j} = \lambda \quad \text{if } k_j > 0, \qquad\qquad v_j \leq \lambda \quad \text{if } k_j = 0.$$

If $\lambda = 0$, the first condition cannot hold, because $v_j e^{-k_j} > 0$, and the second cannot hold, because $v_j > 0$. Hence $\lambda > 0$, and the capacity constraint binds. Solving the conditions type by type gives

$$k_j = \big(\ln v_j - \ln \lambda\big)^+, \qquad\qquad e_j = e^{-k_j} = \min\Big\{1, \frac{\lambda}{v_j}\Big\},$$

and substituting k_j into the binding constraint gives (5).

It remains to show that (5) has exactly one solution. Its left side is continuous in λ, strictly decreasing wherever it is positive, grows without bound as $\lambda \to 0$, and equals zero for $\lambda \geq \max_j v_j$. Thus, it equals $C > 0$ at a unique λ.

Finally, if every type is served, then $(\ln v_j - \ln \lambda)^+ = \ln v_j - \ln \lambda$ for every j, and because $\sum_j \mu_j = 1$, equation (5) becomes

$$\sum_j \mu_j \ln v_j - \ln \lambda = C.$$

Hence $\lambda = e^{-C} \exp\big(\sum_j \mu_j \ln v_j\big) = e^{-C} G$ and $e_j = e^{-C} G / v_j$. □

Proof of Proposition 2. (i) Let $v'_j = (1+\beta) v_j$ for $j \in S$ and $v'_j = v_j$ for $j \notin S$ be the priorities after the change. By Proposition 1, the new allocation is determined by a new threshold λ'.

We guess that

$$\lambda' = \lambda(1+\beta)^m$$

and verify the guess in two steps.

Step 1: every type retains its served or abandoned status. For a type $j \in S$, which is served, $v_j > \lambda$, so

$$v'_j = (1+\beta)v_j > (1+\beta)\lambda > (1+\beta)^m\lambda = \lambda',$$

where the last inequality uses $m < 1$. For a served type $l \notin S$, $e_l = \lambda/v_l$, so the hypothesis $(1+\beta)^m e_l < 1$ is equivalent to $v'_l = v_l > \lambda'$. Finally, an abandoned type satisfies $v'_l = v_l \leq \lambda < \lambda'$ and remains abandoned. The set of served types is therefore still I.

Step 2: the guess uses exactly the available capacity. The capacity used at λ' is

$$\begin{aligned}\sum_{j\in I} \mu_j\big(\ln v'_j - \ln\lambda'\big) &= \sum_{j\in I} \mu_j\big(\ln v_j - \ln\lambda\big) + \mu(S)\ln(1+\beta) - \mu_I\, m\ln(1+\beta)\\ &= C + \mu(S)\ln(1+\beta) - \mu(S)\ln(1+\beta) = C.\end{aligned}$$

The first line uses $\ln v'_j = \ln v_j + \ln(1+\beta)$ for $j \in S$ and $\ln\lambda' = \ln\lambda + m\ln(1+\beta)$, and the second uses $\mu_I m = \mu(S)$. By the uniqueness established in Proposition 1, λ' is the new threshold.

The new error rates are therefore

$$e'_j = \frac{\lambda'}{(1+\beta)v_j} = (1+\beta)^{m-1}e_j \quad \text{for } j \in S, \qquad\qquad e'_l = \frac{\lambda'}{v_l} = (1+\beta)^m e_l \quad \text{for a served } l \notin S,$$

and abandoned types retain an error rate of one.

(ii) The old allocation minimizes the priority-weighted error under the old priorities among all allocations that satisfy the capacity constraint, and the new allocation satisfies

the same constraint. Hence

$$\sum_j \mu_j v_j e'_j \;\geq\; \sum_j \mu_j v_j e_j, \qquad \text{or} \qquad \sum_j \mu_j v_j \big(e'_j - e_j\big) \;\geq\; 0.$$

Separating the terms in S from those outside S and rearranging yields (6). □

Proof of Corollary 1. Because no type enters or leaves service, equation (5) involves only the types in I, and it can be solved for the threshold:

$$\ln \lambda = \frac{1}{\mu_I}\Big(\sum_{j\in I} \mu_j \ln v_j - C\Big).$$

(i) Raising capacity from C to $C + \Delta C$ lowers $\ln \lambda$ by $\Delta C/\mu_I$, so $\lambda' = \lambda e^{-\Delta C/\mu_I}$. The error rate λ/v_j of every served type is therefore multiplied by the same factor $e^{-\Delta C/\mu_I} < 1$, and their ranking does not change.

(ii) Raising the priority of every type in S by the factor $1+\beta$ raises $\sum_{j\in I} \mu_j \ln v_j$ by $\mu(S)\ln(1+\beta)$, and so raises $\ln \lambda$ by $m \ln(1+\beta)$. Together with the added capacity,

$$\lambda' = \lambda (1+\beta)^m e^{-\Delta C/\mu_I}.$$

For a type $j \in S$,

$$e'_j = \frac{\lambda'}{(1+\beta)v_j} = e_j (1+\beta)^{m-1} e^{-\Delta C/\mu_I} < e_j.$$

For a served type $l \notin S$,

$$e'_l = \frac{\lambda'}{v_l} = e_l (1+\beta)^m e^{-\Delta C/\mu_I},$$

which is at most e_l if and only if $(1+\beta)^m \leq e^{\Delta C/\mu_I}$, that is, if and only if

$$\Delta C \;\geq\; \mu_I m \ln(1+\beta) = \mu(S)\ln(1+\beta).$$

Because $m < 1$, at least one served type lies outside S, so the condition is necessary as well

as sufficient. Abandoned types retain an error rate of one.

(iii) We first write the firm's cost in a form that separates the error rates from the list. Let u_{ij} be the share of the firm's inputs that are unlisted inputs of type j. An unlisted input fails at its type's error rate (Section 3.2), so $b_{ij} = u_{ij}e_j$, and the update does not change u_{ij}. Compare two models, A and B, with error rates e_j^A and e_j^B and listed failures a_i^A and a_i^B, in which the firm checks the same type j_1. By (3), the firm's cost under A minus its cost under B is

$$\Delta = c\sum_j u_{ij}\big(e_j^A - e_j^B\big) + c_h\big(a_i^A - a_i^B\big) - (c - c_h)\, s\, \big(e_{j_1}^A - e_{j_1}^B\big).$$

The terms in j_1 combine to $\big[c\, u_{ij_1} - (c - c_h)s\big]\big(e_{j_1}^A - e_{j_1}^B\big)$. Because the firm checks only unlisted inputs of type j_1, $s \le u_{ij_1}$, so the coefficient is at least $c\, s - (c - c_h)s = c_h s > 0$. Every coefficient in Δ is therefore positive, and

(a) if $e_j^A \le e_j^B$ for every j and $a_i^A \le a_i^B$, then $\Delta \le 0$;

(b) if $e_j^A \ge e_j^B$ for every j, with strict inequality for a type with $u_{ij} > 0$, and $a_i^A \ge a_i^B$, then $\Delta > 0$.

For the first claim, let A be the model after the update and B the model after an update with the same churn that changes neither capacity nor priorities, under which every error rate stays at e_j. By part (ii), $e_j^A \le e_j$ for every j. The reporter's listed failures are fixed under A. Any other listed failure of type j still fails with probability $(1 - \chi)\min\{1, e_j^A/e_j\} + \chi e_j^A$ under A and with probability $(1 - \chi) + \chi e_j$ under B, and the first is at most the second because $e_j^A \le e_j$. Hence $a_i^A \le a_i^B$, and by (a), no firm's cost is higher under A.

For the second claim, let A be the model after the report at fixed capacity and B the model after an update with the same churn and no change in priority, under which every error rate stays at e_j. Consider a firm that uses a served type outside S and no type in S. By Proposition 2(i), every served error rate outside S is multiplied by $(1 + \beta)^m > 1$ and abandoned types retain an error rate of one, so $e_j^A \ge e_j^B$ for every type the firm uses, with strict inequality for its served types. A listed failure of type j still fails with probability

$(1-\chi)+\chi e_j^A$ under A, because its error rate did not fall, and with probability $(1-\chi)+\chi e_j$ under B, so $a_i^A \geq a_i^B$. A served type l that the firm uses has $u_{il} > 0$, because $e_l < 1$, so some of the firm's inputs of type l succeed and are therefore unlisted. By (b), the firm's cost is higher after the report. □

Proof of Lemma 1. We show that tasks with too little headroom and tasks with too much headroom are both unprofitable, with bounds that do not depend on C.

Too little headroom. Because $r_C(h) \leq \bar{r}$,

$$\Pi_C(h) \leq q(h)\,(\bar{r}+c) - c - g_0,$$

which is at most zero whenever $q(h) \leq (c+g_0)/(\bar{r}+c)$. This threshold lies in $(0,1)$ because $0 < g_0 < \bar{r}$. Because q is continuous and rises strictly from zero to one, the headroom

$$\underline{h} = q^{-1}\Big(\frac{c+g_0}{\bar{r}+c}\Big)$$

is finite and does not depend on C, and $\Pi_C(h) \leq 0$ for every $h \leq \underline{h}$.

Too much headroom. Because $q(h) \leq 1$,

$$\Pi_C(h) \leq \max\{r_C(h), 0\} - g_0.$$

Because $\sup_{C \geq C_0} r_C(h) \to 0$ as $h \to \infty$, there is a finite $\bar{h}$, which does not depend on C, such that $r_C(h) \leq g_0$, and hence $\Pi_C(h) \leq 0$, for every $h \geq \bar{h}$.

Conclusion. By assumption, $\Pi_C(h) > 0$ for some h, which must lie strictly between $\underline{h}$ and $\bar{h}$. On the interval $[\underline{h}, \bar{h}]$, the continuous function Π_C attains a maximum, which is positive, and every point outside the interval has $\Pi_C \leq 0$. Every most profitable task therefore has headroom in $[\underline{h}, \bar{h}]$, and its difficulty $C-h$ lies in $[C-\bar{h},\, C-\underline{h}]$, which stays within $\max\{|\underline{h}|, |\bar{h}|\}$ of C and so grows without bound as C grows. □

Proof of Proposition 3. How an update changes which inputs fail. The update of Section 3.3

can be represented as follows. Each input of type j has a rank drawn from the uniform distribution on $[0,1]$, independently across inputs, and the input fails if its rank is below its type's error rate. The update keeps the rank with probability $1-\chi$ and redraws it with probability χ. Then

- a kept input that failed before has a rank that is uniform on $[0, e_j)$, so it continues to fail with probability $\min\{1, e'_j/e_j\}$;
- a kept input that succeeded before has a rank that is uniform on $[e_j, 1]$, so it now fails with probability $(e'_j - e_j)^+/(1-e_j)$;
- a redrawn input fails with probability e'_j, whatever happened before.

(i) The reporter's listed failures are fixed by the definition of a report. Any other listed failure of type j continues to fail with probability

$$(1-\chi)\min\left\{1, \frac{e'_j}{e_j}\right\} + \chi e'_j \;\leq\; 1 - \chi + \chi e'_j,$$

which is below one when $\chi > 0$ and $e'_j < 1$. An input of type j that the firm previously observed to succeed now fails with probability

$$(1-\chi)\frac{(e'_j - e_j)^+}{1-e_j} + \chi e'_j,$$

which is positive when $\chi > 0$ and $e'_j > 0$.

Map quality. Map quality is measured on the inputs that the firm has not yet seen. Their ranks do not depend on the firm's experience, so they are uniform before the update and, under either branch, after it. An unseen input of type j therefore fails after the update with probability e'_j, independently across inputs.

Consider a firm that uses two types, j and l, in shares w_j and w_l, with e'_j and e'_l strictly between zero and one. Draw one unseen input that fails and one that succeeds. The failing input is of type j with probability $w_j e'_j/P$, where $P = w_j e'_j + w_l e'_l$, and the succeeding input

is of type j with probability $w_j(1-e'_j)/Q$, where $Q = w_j(1-e'_j) + w_l(1-e'_l)$. A map that ranks j above l ranks the pair correctly when the failing input is of type j and the succeeding input is of type l, and it ties, which counts as one half, when the two inputs are of the same type. Its AUC is therefore

$$\mathrm{AUC}_{j>l} = \frac{1}{PQ}\Big[w_j e'_j\, w_l(1-e'_l) + \tfrac{1}{2}w_j^2 e'_j(1-e'_j) + \tfrac{1}{2}w_l^2 e'_l(1-e'_l)\Big].$$

The AUC of the map that ranks l above j is the same, except that the first term is $w_l e'_l\, w_j(1-e'_j)$. The difference is

$$\mathrm{AUC}_{j>l} - \mathrm{AUC}_{l>j} = \frac{w_j w_l}{PQ}\Big[e'_j(1-e'_l) - e'_l(1-e'_j)\Big] = \frac{w_j w_l}{PQ}\big(e'_j - e'_l\big).$$

The map that ranks first the type with the higher new error rate therefore has the higher AUC, and the two maps have the same AUC when the two rates are equal.

(ii) Every type that the firm uses is served and lies outside S, so by Proposition 2(i) each of their error rates is multiplied by the same factor $(1+\beta)^m$ and stays below one. Their ranking does not change, so the firm's map stays correct. Because $e_l = \lambda/v_l$ for each of these types, the firm's error rate rises from $\sum_l w_{il}\lambda/v_l = \lambda\rho_i$ to $(1+\beta)^m\lambda\rho_i$, an increase of $\lambda\rho_i\big[(1+\beta)^m - 1\big]$.

(iii) Because $e_j > e_l$ and both types are served, $\lambda/v_j > \lambda/v_l$, so $v_j < v_l$, and the firm's map, which is correct before the update, ranks j first. After the report,

$$e'_j = \frac{\lambda'}{(1+\beta)v_j}, \qquad e'_l = \frac{\lambda'}{v_l},$$

so

$$e'_j < e'_l \quad\Longleftrightarrow\quad v_l < (1+\beta)v_j.$$

When this condition holds, a relearned map ranks l first, and by the comparison above the old map has a strictly lower AUC. When it fails, $e'_j \geq e'_l$, so the old map does not go stale,

and if $e'_j = e'_l$ the two maps have the same AUC. The new error rates depend only on S, not on who made the report, so the result holds whether the firm or a competitor made the report. Finally, the firm checks type j under the old map and type l under a relearned map, and nothing else in (3) depends on the map, so its cost with the old map exceeds its cost with a relearned map by $(c - c_h)\, s\, (e'_l - e'_j)$. □

Proof of Proposition 4. By Proposition 2(i), the update multiplies the error rate of every type in S by $(1+\beta)^{m-1}$ and that of every type in N by $(1+\beta)^m$, and it leaves the error rate of every abandoned type at one. For $j \in S$ and $l \in N$, let

$$\Delta_j = e_j - e'_j = \left[1 - (1+\beta)^{m-1}\right] e_j, \qquad \Delta_l = e'_l - e_l = \left[(1+\beta)^m - 1\right] e_l$$

be the fall in the error rate of j and the rise in the error rate of l. Both are positive, because $\beta > 0$ and $0 < m < 1$. Let

$$T(\beta) = \frac{(1+\beta)^m - 1}{1 - (1+\beta)^{m-1}} > 0, \qquad \text{so that} \qquad R(\beta) = \frac{1-m}{m}\, T(\beta).$$

Dividing Δ_l by Δ_j,

$$\frac{\Delta_l}{\Delta_j} = \frac{e_l}{e_j}\, T(\beta) = \frac{v_j}{v_l}\, T(\beta),$$

where the last step uses $e_j = \lambda / v_j$ and $e_l = \lambda / v_l$, which hold for served types by Proposition 1. In particular, $\Delta_l = T(\beta)\Delta_j$ when $e_j = e_l$.

(i) Each input that the firm has not yet seen fails at its type's current error rate, both before and after the update (proof of Proposition 3). Every type that the firm uses is served and lies outside S, so its error rate is multiplied by $g = (1+\beta)^m$, and it stays below one by the conditions of Proposition 2. All the firm's error rates are multiplied by the same factor, so their ranking does not change, and the firm checks the same type l_1, the one its map ranks first, before and after the update. By the same accounting as in (ii) below, its expected cost

on its unseen inputs is

$$c\Big[\sum_{l} n_{il}e_l - \theta s\, e_{l_1}\Big]$$

before the update, and the same expression with every e_l replaced by ge_l after it, which is g times the cost before.

(ii) Each input that the firm has not yet seen fails at its type's current error rate, both before and after the update (proof of Proposition 3). The firm's map is correct before the update, so it ranks j first, and beyond its list the firm checks a share s of its inputs from type j, which are inputs it has not yet seen. With its old map, it checks the same type after the update. Of its unseen inputs of type j, a share $n_{ij} - s$ is not checked, and each error there costs c, while a share s is checked, and each error there costs $c_h = (1-\theta)c$. Each error on its unseen inputs of type l costs c. By (3), the firm's expected cost on its unseen inputs is therefore

$$c\big[(n_{ij} - s)e_j + (1-\theta)s\, e_j + n_{il}e_l\big] = c\big[(n_{ij} - \theta s)e_j + n_{il}e_l\big]$$

before the update, and the same expression with e'_j and e'_l after it. The cost before minus the cost after is

$$c\big[(n_{ij} - \theta s)\Delta_j - n_{il}\Delta_l\big].$$

This is positive if and only if $(n_{ij} - \theta s)\Delta_j > n_{il}\Delta_l$, which is the condition in (ii). Because $\Delta_j > 0$, dividing by Δ_j and using the ratio above gives the condition in terms of the priorities,

$$n_{ij} - \theta s > n_{il}\,\frac{v_j}{v_l}\,T(\beta).$$

(iii) Abandoned types keep the error rate one, so only served types change:

$$L' - L = -\sum_{j\in S} \mu_j\sigma_j\Delta_j + \sum_{l\in N} \mu_l\sigma_l\Delta_l.$$

Substituting Δ_j and Δ_l, with $e_j = \lambda/v_j$, and using $(1+\beta)^m - 1 = T(\beta)\big[1-(1+\beta)^{m-1}\big]$,

$$L' - L = \lambda\big[1-(1+\beta)^{m-1}\big]\Big[T(\beta)\sum_{l\in N}\mu_l\frac{\sigma_l}{v_l} - \sum_{j\in S}\mu_j\frac{\sigma_j}{v_j}\Big].$$

Because $\lambda > 0$ and $1-(1+\beta)^{m-1} > 0$, $L' < L$ if and only if the sum over S exceeds $T(\beta)$ times the sum over N. On a served type, $\sigma_j/v_j = \sigma_j e_j/\lambda$, so the sum over S is $\mu(S)\bar{\ell}_S/\lambda$ and the sum over N is $\mu(N)\bar{\ell}_N/\lambda$. The set N is nonempty, because $m < 1$. The served types are those in S and in N, so $\mu(S) + \mu(N) = \mu_I$ and $m = \mu(S)/\mu_I$, which gives $\mu(N)/\mu(S) = (1-m)/m$. The condition is therefore

$$\bar{\ell}_S > \frac{\mu(N)}{\mu(S)}\,T(\beta)\,\bar{\ell}_N = R(\beta)\,\bar{\ell}_N.$$

It remains to show the three properties of $R(\beta)$. Let $x = \ln(1+\beta) > 0$. Then

$$R(\beta) = \frac{\phi(x)}{\psi(x)}, \qquad \phi(x) = (1-m)\big(e^{mx}-1\big), \qquad \psi(x) = m\big(1-e^{-(1-m)x}\big).$$

Both ϕ and ψ equal zero at $x = 0$, and both have the derivative $\phi'(0) = \psi'(0) = m(1-m)$ there.

- *It exceeds one.* For $x > 0$, $\phi'(x) - \psi'(x) = m(1-m)\big(e^{mx} - e^{-(1-m)x}\big) > 0$, and $\phi(0) - \psi(0) = 0$, so $\phi(x) > \psi(x) > 0$.

- *It rises with β.* $\phi''(x) = m^2(1-m)e^{mx} > 0$, so ϕ is strictly convex with $\phi(0) = 0$, and the slope of its chord from the origin, $\phi(x)/x$, strictly rises with x. $\psi''(x) = -m(1-m)^2e^{-(1-m)x} < 0$, so ψ is strictly concave with $\psi(0) = 0$, and $\psi(x)/x$ strictly falls with x. Their ratio, $\phi(x)/\psi(x) = \big(\phi(x)/x\big)\big/\big(\psi(x)/x\big)$, therefore strictly rises with x, and hence with β.

- *It tends to one.* As $x \to 0$, $\phi(x)/x \to \phi'(0) = m(1-m)$ and $\psi(x)/x \to \psi'(0) = m(1-m)$, so the ratio tends to one.

(iv) Suppose $\sigma_j = \gamma v_j$, so that $\sigma_j / v_j = \gamma$, for every served type. Then the sum over S is $\gamma\mu(S)$ and the sum over N is $\gamma\mu(N)$, and by the expression in (iii),

$$L' - L = \lambda\gamma\big[1 - (1+\beta)^{m-1}\big]\mu(S)\big[R(\beta) - 1\big] > 0,$$

because $R(\beta)$ exceeds one. □