

Intelligence as Capital

Learning, Reusable Software, and the Cost of Decisions

October 12, 2026

Abstract

Must a business purchase advanced artificial intelligence each time it makes a decision, or can an initial investment create software that handles recurring decisions inexpensively? We study this question through a sequence of simulated inventory, allocation, and business-management experiments. When rules are supplied, even inexpensive models often build programs that make optimal decisions on tested cases. In a larger experiment with initially unknown conditions, experience improves performance, while expensive teaching adds little at the declared economic tolerance. In harder simulated businesses, advanced models build substantially better programs in the observed sample. Four such programs subsequently operate without model calls on fresh scenarios, achieving higher average operating performance than a frontier model making daily decisions. The final experiment is too small for its conservative statistical test to establish preserved average quality. Across the studies, the clearest finding is that AI can create reusable decision capital: intelligence purchased during development need not be purchased again during operation. The evidence does not establish that experience is necessary for this conversion, that a smaller model can take over an initially frontier-dependent task, or that the result extends to arbitrary AI applications.

Keywords: artificial intelligence; learning by doing; intangible capital; stochastic control; automation.

JEL codes: D83, O33, C63.

1 Introduction

The price of a decision depends on how it is produced. A firm can repeatedly hire an expert, train a less expensive operator, or invest in a procedure that handles future cases. Artificial intelligence permits all three arrangements. A capable language model may choose an action directly, teach another model, or write a program that makes subsequent choices without further model inference. These arrangements have different development costs, operating costs, and requirements for reuse.

We investigate whether experience with a fixed task makes competent decision-making inexpensive. The motivating hypothesis is stronger than ordinary improvement with practice: a task that initially benefits from expensive intelligence might eventually be handled cheaply because the system preserves what it has learned. Testing this hypothesis requires separating the intelligence used to *build* a decision procedure from the intelligence required to *operate* it. It also requires an initial capability gap. A cheap procedure that works immediately demonstrates inexpensive construction, not a transition caused by learning.

This paper synthesizes the complete experimental sequence rather than selecting a single favorable outcome. The studies progressively change what the model can do and what it knows: from choosing

inventory orders directly, to writing reusable solvers, to learning an unknown member of a supplied model class, to managing businesses whose response mechanisms are not disclosed. The final studies explicitly permit code, memory, tools, and changes to the operating architecture. Each extension has its own protocol and evidence; they are not replications of one common estimand.

Three findings organize the evidence. First, allowing executable programs changes the result markedly. In small inventory problems, all three model tiers construct programs with zero measured decision loss relative to an exact optimizer. These programs also transfer to a prespecified larger setting. In scheduling, invoice matching, and event processing, both inexpensive and frontier models construct successful reusable programs; changes to the specification require repair. These findings establish the usefulness of software construction but often leave little initial advantage for frontier intelligence to explain.

Second, learning is useful without necessarily requiring an expensive teacher. The largest fresh-task confirmation includes 1,024 independently sampled tasks from four finite control families. Additional experience and construction effort reduce the gap to an informed benchmark. Frontier teaching provides no economically material improvement over the own-experience learner at the declared tolerance, while increasing mean learning API expenditure from about \$0.008 to \$0.220. Much of the improvement is learning which supplied mechanism governs the task. This is valuable information acquisition, but a limited test of discovering an unknown decision mechanism.

Third, a richer business environment produces an observed frontier construction advantage and a particularly clear distinction between construction and operation. In a twelve-business main panel, frontier-built controllers substantially outperform inexpensive construction at the initial checkpoint. In a separate two-business development exercise, all four frontier systems eventually produce programs that require no operating model. A fresh evaluation of these unchanged programs contains 192 completed episodes. Software achieves higher sample-average performance than the initial frontier operator while eliminating approximately \$4.96 of recurring model charges per simulated sixty-day business episode. Its conservative, predeclared confidence interval remains too wide to establish preserved average quality. That limitation was foreseeable from the design, not a surprising consequence of an unusually variable observed sample.

The economic contribution is a distinction supported across settings: advanced intelligence can be an input into a durable decision asset rather than a necessary recurring input. The stronger causal proposition remains unresolved. The studies do not establish that frontier intelligence was initially necessary among feasible alternatives, that accumulated observations rather than additional development computation caused the conversion, or that a smaller operating model successfully inherited a frontier system. These distinctions determine what the evidence says about demand for AI services.

Related literature. The connection to learning by doing is direct: experience can improve productive capability (Arrow, 1962b). Here, however, experience is combined with deliberate model-assisted development. The resulting code and procedures can be reused, linking the experiment to investment in knowledge (Arrow, 1962a) and complementary intangible capital (Brynjolfsson et al., 2021). Deployment savings are not equivalent to lower total costs when learning, validation, engineering, and replacement are expensive.

Control theory supplies both the language and serious conventional alternatives. Recht (2019) emphasizes what structured control problems reveal about learning and planning; Mania et al. (2018) show that simple linear policies and random search can be competitive on selected continuous-control benchmarks. These results motivate testing simple controllers before interpreting a model advantage

as a necessity. They do not establish that simple policies suffice for arbitrary business problems. Our transfer experiments concern persisted code, prompts, memory, and procedures; they are distinct from training smaller neural-network weights to reproduce a teacher’s policy (Rusu et al., 2015).

2 A Minimal Framework

Let θ denote the fixed rules of one task: how actions and conditions generate outcomes, what can be observed, and how performance is valued. It is a mechanism, not the task’s current condition. For example, θ may specify how production effort creates wear and how maintenance repairs it; current wear changes as the business operates. A compact representation is

$$x_{t+1} = \theta(x_t, a_t, \varepsilon_{t+1}), \quad a_t = g_{m, S_N}(h_t). \quad (1)$$

Here x_t is the possibly partly hidden business condition, ε_{t+1} is a random disturbance, and h_t is the information actually available at the decision. The controller g uses operating model m and transferable state S_N accumulated during N experience episodes. That state may contain software, prompts, fitted parameters, memory, or procedures. A controller implemented entirely in ordinary software has no operating model. Observation and payoff rules are understood to be part of the task specification; equation (1) does not require that the controller observe x_t or know θ .

Let $J_\theta(g)$ be expected operating performance, with larger values preferred. There are three distinct comparisons: replacing repeated inference with a program; improving an affordable learner with frontier lessons; and changing only the operating model of an already-developed system. For the last comparison, the natural quality requirement is

$$J_\theta(g_{\text{cheap}, S_N}) \geq J_\theta(g_{\text{frontier}, S_N}) - \delta, \quad (2)$$

where δ is a tolerance fixed before evaluation. Comparing cheap operation with S_0 and S_N identifies the contribution of the saved state only to the extent that other changes are controlled. A zero-call program is a separate endpoint: changing its nominal model binding changes nothing.

If acquisition costs A , recurring execution costs c , and the procedure remains useful for M deployments, average computing expenditure is

$$\overline{C}(M) = A/M + c. \quad (3)$$

This accounting identity makes reuse central. It does not value decision errors or establish that the task lasts long enough to repay investment. Acquisition should include learning, testing, failures, and transfer; engineering and maintenance need separate treatment when unmeasured. No general compression theorem is assumed.

3 Experimental Sequence and Measurement

Table 1 summarizes the environments; Table 2 separates the interventions. The exact model identifiers and settings are retained in the accompanying protocols. We use their tier labels: Luna for the inexpensive model, Sol for an intermediate model where included, and Astra for the frontier model. These are the recorded hosted-model routes, not guarantees of immutable underlying weights. All learning in this program changes the surrounding system; model weights are held fixed.

The meaning of quality changes with the available reference. In the known-rule studies, *regret* is the expected payoff lost relative to an exact optimizer with the same information. Inventory

Table 1: Environments and the information supplied to controllers

Environment	Decisions and uncertainty	Information and benchmark
Inventory	Two products, initially three periods; demand and delivery shocks; independent or shared ordering capacity. Extensions vary size and horizon.	Dynamics supplied. Exact dynamic programming evaluates actions under the same information.
Reuse tasks	Scheduling, invoice matching, and event-log processing; random instances, wider parameters, and changed contracts.	Rules supplied; exact task-specific references. Not all tasks require sequential control.
Finite-model learning	Contextual, fully observed sequential, interacting, and partially observed control; horizons at most three.	At most 16 candidate mechanisms and a prior supplied. Persistent candidate initially unknown. Exact information-matched evaluation.
Unknown-mechanism businesses	Six production units, sixty days; effort, wear, delayed maintenance, noisy observation, and interactions in groups of one, three, or six.	Public objective and interface supplied; response mechanism withheld. Frontier and conventional comparators, with no exact optimum.
Vending adaptation	Business-management simulator and ninety-day conventional rollouts.	Model experiment not completed; no AI treatment effect reported.

Notes: Each study retains its own generated-task distribution and payoff scale. Random disturbances continue during evaluation. A fixed mechanism does not imply an unchanging condition or deterministic outcomes.

Table 2: What is transferred, and which question it tests

Approach	Intervention	Evidence and interpretation
Direct model decisions	Change model tier, reasoning effort, or text memory.	Inventory: capability matters; more effort and text learning are not uniformly beneficial.
Model to software	Model writes an executable controller; later calls are unnecessary.	Inventory, reuse tasks, and businesses: observed successful construction, including fresh software validation.
Frontier to cheap learner	Frontier provides lessons; cheap model develops its own controller.	Finite-model confirmation and business curricula: distinct from transferring a finished system.
Frontier to cheap operator	Freeze the entire system; change its operating model.	Planned small handoff reached the zero-call branch first. Smaller-model substitution was not established.
Conventional control	Handwritten exact planners, feedback rules, or fitted planners.	Exact planning solves small known problems; feedback remains a useful business benchmark.

Notes: Construction, teaching, and operation have separate costs. A frontier-built program can be cheaper to operate than a cheap-model agent.

evaluation sums the optimizer’s pre-shock disadvantage of each chosen action along the controller’s own trajectory. In the finite-model study, exact policy evaluation integrates over possible trajectories. Zero regret therefore means equality to the relevant information-matched reference; it does not mean no randomness or, for sampled inventory paths, optimality at every possible state.

In the business studies, quality is net completed-work value after effort, maintenance, rejected-work, carrying, and terminal costs. One completed unit contributes one score unit before costs. Scores are neither dollars nor percentages. Without an exact optimizer, a favorable comparison to the frontier agent cannot establish near-optimality.

Frozen evaluations follow each controller’s own actions. Shared external shocks create paired comparisons without forcing policies onto another policy’s trajectory. Fresh-task inference treats independently sampled tasks as the units; repeated episodes and construction repeats do not create additional independent tasks. The final two-business validation instead conditions on those fixed businesses and programs and uses independent scenario batches. Failed artifacts and declared fallbacks remain in performance accounting.

4 Known Rules: Direct Reasoning, Construction, and Reuse

4.1 Direct decisions and deeper text learning

The original inventory experiment contains 1,728 paid evaluation episodes across 72 cells, plus 192 baseline episodes. Actors return an order through one model request per decision. They have a public specification, legal actions, a bounded memo, and retrieved examples, but no executable calculator or reusable program. The fixed Sol-high builder updates the text harness. Independent capacity permits separate orders; shared capacity couples orders through a common resource constraint.

Astra has zero measured regret in 22 of 26 reported cells. Sol’s cell means range from zero to 0.571, and Luna’s from 0.219 to 3.704. Extra reasoning is useful in some comparisons but not monotonically so. For example, Luna’s final learned shared-capacity mean falls from 3.098 at medium effort to 0.856 at high effort. Learning is also uneven; retrieved examples can change even when the memo is retained. These are comparisons among particular model-and-harness configurations, not a general frontier scaling law.

A separate extension develops two learning lineages through 3, 8, 24, and 64 experiences, compares fourteen candidates, and freezes Luna-high at the final checkpoint. Fresh confirmation uses 2,000 episodes in each capacity condition. The independent condition has mean regret 0.6916 and a valid upper limit of 1.0681. The shared condition has mean 1.2996 and running-minimum upper limit 1.5228. Neither establishes the declared one-unit tolerance. Ninety-six invalid orders among 12,000 decisions retain their prescribed zero-order fallbacks. The optional longer-horizon text experiment was not run.

4.2 Executable programs and structural transfer

Allowing code changes the production method. With four development rounds, all three model tiers construct dynamic-programming controllers. Each frozen program has zero exact regret and no fallback on its 32 pilot episodes. Construction API charges are approximately \$0.0035, \$0.0481, and \$0.3394 for Luna, Sol, and Astra; execution requires roughly 0.10 CPU seconds per episode. These are single construction lineages, not estimates of population build reliability.

The same frozen programs, without repair, again have zero measured regret on 64 transfer cases

each, with horizon increased to five and capacity to five. A separate structural study compares planners over eighteen inventory configurations. Across 81 solver–configuration cells, 46 pass and 35 fail, but the full joint dynamic program passes all eighteen configurations. On 48 held-out cases it has zero regret and takes about 0.099 CPU seconds; the proposed router has regret 0.00225 per product-period and takes about 0.101 seconds. Adding interaction features does not improve its held-out predictions over size-only features. The experiment did not find the boundary at which inexpensive full planning breaks down.

4.3 Reuse, changes in specification, and repair

The cross-domain pilot studies scheduling, invoice matching, and event logs. Luna and Astra each receive two independent construction attempts and corresponding repair opportunities per family. Development permits execution feedback; direct comparators can use code tools but cannot preserve a solver across cases. Baseline tasks, wider parameters, and a changed contract distinguish within-specification reuse from obsolete software.

All 864 intended reusable evaluations, 72 final direct evaluations, and 216 conventional evaluations achieve exact results. Unrepaired programs fail every quality gate after the contract changes; repair restores performance. The result is successful reuse within a specification, not immunity to changes in the task. Both model tiers succeed, so this pilot does not establish a frontier construction advantage.

Measured API charges and priced execution imply repayment after two deployments across the first two phases or three across the full lifecycle. The absolute savings are small: at 3,072 deployments, projected savings are only about \$1.92–\$2.56. Adding \$50 of engineering raises repayment to roughly 60,000–80,000 deployments. This is an important economic qualification: a large proportional decline in inference cost can have little value when the original task is already very cheap.

5 Learning Within a Supplied Model Class

5.1 Design and main learning curves

The four-family study asks whether experience helps an affordable learner and whether frontier teaching accelerates that improvement. Tasks vary in reusable structure, observability, and interaction, but supply a small list of candidate mechanisms and a prior. The unknown persistent mechanism is one member of that list. Partially observed tasks additionally contain changing hidden conditions. Discovering which candidate applies is therefore different from inventing an unrestricted model of the world.

An eight-preset pilot precedes a main panel of sixteen tasks, two independent learning repeats, and checkpoints at 0, 8, 32, and 128 episodes. Comparisons include own experience, cheap teaching, frontier teaching, a spending control, frontier construction, repeated model inference, and conventional procedures. Models can write reusable code and preserve memory. Teacher comparisons match inherited code and underlying histories where required; teachers provide examples and lessons rather than simply handing over a completed solver. More experience also entails additional development computation.

Figure 1 shows the principal learning curves. Cheap initial construction already meets the quality criterion in both repeats for fifteen of sixteen main tasks. Control quality is not monotone with experience or teacher capability. At 128 episodes, mean control regret is 0.0307 with own experience, 0.0360 with cheap teaching, and 0.0215 with frontier teaching. Their relative ordering changes at

earlier checkpoints.

The gap to a benchmark told the true persistent mechanism decomposes into the value of that information and the controller’s remaining decision loss. For the own learner, the total gap falls from 0.8288 to 0.0408 between zero and 128 episodes, while the information component falls from 0.7629 to 0.0101. Most of the improvement concerns knowing which world the system occupies. The agent still has to use that information, but this is weaker evidence of learning a fundamentally new decision procedure.

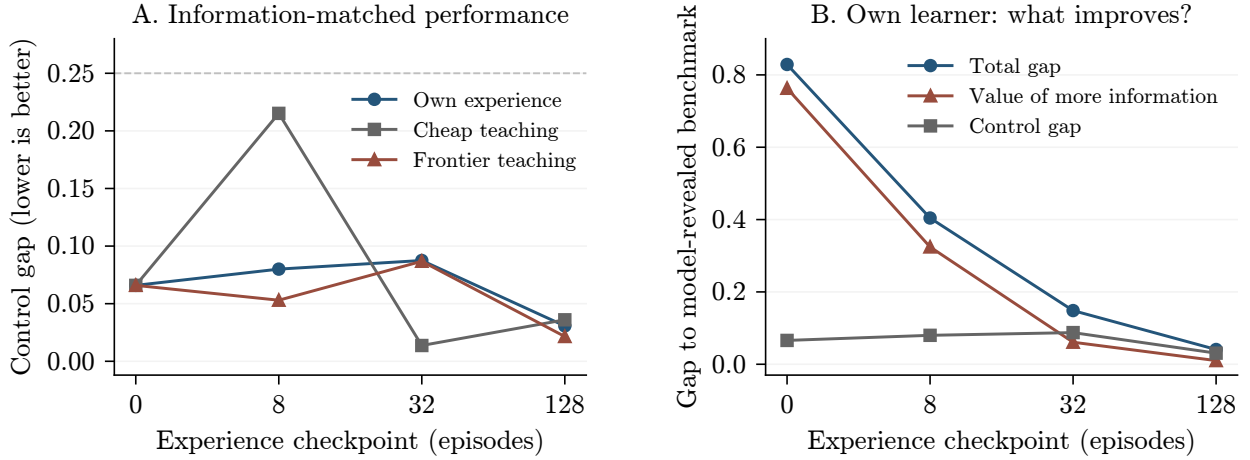


Figure 1: Learning quality and learning which mechanism applies

Notes: Descriptive main-study means: sixteen tasks and two repeats. Left: loss against an exact optimizer with the same information; the dashed line marks 0.25. Right: own-experience learner’s total gap to a model-revealed benchmark, separated into information value and control loss. Curves do not carry a simultaneous confidence guarantee.

Eight separate tasks with held-out structural combinations provide a small transfer check. Mean regret falls from 0.8810 at zero experience to 0.0447 after eight episodes with own experience, versus 0.0366 with frontier teaching. One poor initial controller accounts for 93.8% of initial total regret. This panel is descriptive and remains separate from confirmation. The main study’s interaction range is also limited: only one of four compositional tasks has strong coupling.

5.2 Independent confirmation on 1,024 tasks

The main-selected procedure is own learning after eight episodes. Selection among positive learning checkpoints excludes zero experience; that restriction is retained when interpreting the result. A frozen learning procedure then constructs controllers on 1,024 fresh independently sampled tasks, with one new learning run per task. The four families are sampled uniformly, not forced to have equal realized counts. Table 3 reports every primary endpoint and retains construction failures.

The own-experience controller’s mean regret is 0.1159. Its valid upper limit is 0.2586, just above the predeclared raw tolerance of 0.25. Its normalized upper limit, 0.0320, passes the separate 0.05 criterion, but both are required. Thus selected-policy near-optimality remains inconclusive. The frontier-taught endpoint has an upper limit of 0.2410, but selecting it after confirmation would change the procedure under test.

The teaching advantage is 0.0073, with simultaneous interval $[-0.1477, 0.1684]$. This entire

Table 3: Fresh-task confirmation: performance, failures, and learning expense

Experience	Procedure	Mean regret	Exact-optimal	Failures	API learning
0	Own experience	0.1995	914	100	\$0.0040
8	Own experience	0.1159	929	16	\$0.0080
8	Frontier teaching	0.1086	931	16	\$0.2195

Notes: Each row covers the same 1,024 task draws. Exact-optimal counts refer to expected performance with the controller’s information. API learning costs are means per task, include required ancestors and recorded failures, and overlap across counterfactual procedures; they must not be added as project spending. All three deploy without model calls.

interval lies within the declared ± 0.25 equivalence region: at that tolerance, frontier teaching and own learning are practically equivalent. This is not a comparison proving equivalence at every smaller tolerance or between all teacher types. The predeclared experience advantage is 0.7271, with interval [0.5496, 0.9067]. It measures the reduction in the *total model-revealed gap*, including information acquisition, and combines new experience with further construction. It must not be reported as a 0.7271 reduction in control regret alone.

The five statements have at least 95% simultaneous coverage across the declared looks under the protocol’s sampling assumptions. Exact evaluation removes trajectory-sampling error, but random tasks and controller construction still produce uncertainty. For the selected procedure, 1,008 of 1,024 artifacts meet operational affordability criteria; mean measured execution is 0.0875 CPU seconds and deployment uses no model calls. Mean learning API cost is approximately \$0.0080, compared with \$0.2195 under frontier teaching. Learning is useful, but these results show little need for frontier teaching within the tested model classes.

6 Unknown Mechanisms and the Value of Construction

6.1 From a supplied menu to operating a business

The next environment removes the supplied list of transition models. Each business operates six production units for sixty days. Actions allocate production and maintenance effort; production creates wear, maintenance can have delayed effects, and productivity responds nonlinearly to conditions. Units share effort constraints and may interact within groups of one, three, or six. The controller observes pending work, a noisy wear signal, and the public consequences of previous actions. The objective and feasible actions are supplied, while the fixed response mechanism is withheld.

An initial screen includes six businesses and two independent builds per model tier: 24 artifacts. Each receives one live episode permitting limited advice and sixteen separate advice-free episodes. With independent units, cheap, frontier, and conventional feedback mean scores are 5.73, 47.78, and 33.83. With groups of three they are -88.05 , -47.36 , and 32.79; with a group of six, -4.28 , 15.65, and 28.13. The apparent frontier advantage is thus not sufficient to establish competence: conventional feedback is better in the two coupled settings. Ten frontier construction responses hit the output cap. The original feasibility gate ends this phase before later learning checkpoints; it supplies a development screen, not a confirmed transition.

6.2 An intermediate curriculum and an information diagnostic

A subsequent software-construction curriculum covers three independent businesses with separate units, two repeats, and checkpoints 0, 2, 8, and 32. Here zero denotes one starter episode before additional experience. Five core learning curves finish before the project changes scope; the spending-control expansion, planned confirmation, and structural holdout do not. Table 4 preserves the completed evidence. Affordable revisions deteriorate on average; frontier construction remains stronger. The no-new-history arm also deteriorates, so the comparison cannot be summarized as evidence that acquiring more data itself caused the decline.

Table 4: Intermediate software curriculum: completed core only

Procedure	0	2	8	32
Own experience	7.09	-0.78	-13.19	-8.91
Cheap teaching	7.09	-25.65	-28.81	-23.58
Frontier teaching	7.09	-10.25	-20.58	-27.59
No new history	7.09	-8.35	-22.57	-31.75
Frontier construction	49.09	59.48	55.58	54.11

Notes: Mean live net work-value scores, higher is better; three businesses and two repeats. Live operation permits at most one same-tier advice call. Sixteen advice-free replays per artifact are a separate endpoint. The incomplete spending control is excluded; these are developmental means, not completed fresh confirmation.

A separate disclosure diagnostic builds 24 artifacts on two development businesses, with two repeats, holding nine collected episodes fixed. Supplying transition structure, noise laws, and interaction information raises cheap-model mean live performance from -21.37 to 10.39 ; additionally supplying all fixed coefficients yields -2.92 . Frontier means are 54.66 , 57.29 , and 57.59 . Full mechanism knowledge does not reveal the current hidden condition or provide an optimal solver. This small comparison suggests that the observed construction gap is not simply eliminated by disclosing the rules. It neither identifies a unique source of cheap-model failure nor supports broad population inference.

6.3 The twelve-business whole-system study

The larger main study permits learning the entire agent, including executable software and optional model use. Twelve independently generated businesses are balanced across the three group sizes, with two repeats and checkpoints 0, 2, 8, and 32. Five longitudinal procedures are compared, alongside repeated inference and conventional controllers. Each frozen artifact receives four protected scenarios per repeat, always under its own actions. Additional development trials count as experience and expense. Matched-input teaching branches address lesson quality separately from the different histories generated by longitudinal arms.

Figure 2 reports the main curves. Initial cheap construction averages -52.07 , versus 54.32 for frontier construction. At checkpoint 32, own learning reaches 22.79 and frontier-taught cheap construction reaches 32.59 ; frontier construction reaches 58.39 . The conventional feedback rule averages 40.05 and repeated initial frontier inference 49.86 . Frontier-taught cheap construction is selected by the declared fallback rule, because no candidate satisfies all main targets. None of the 24 cheap learning repeats in each arm achieves a sustained observed target crossing across later checkpoints.

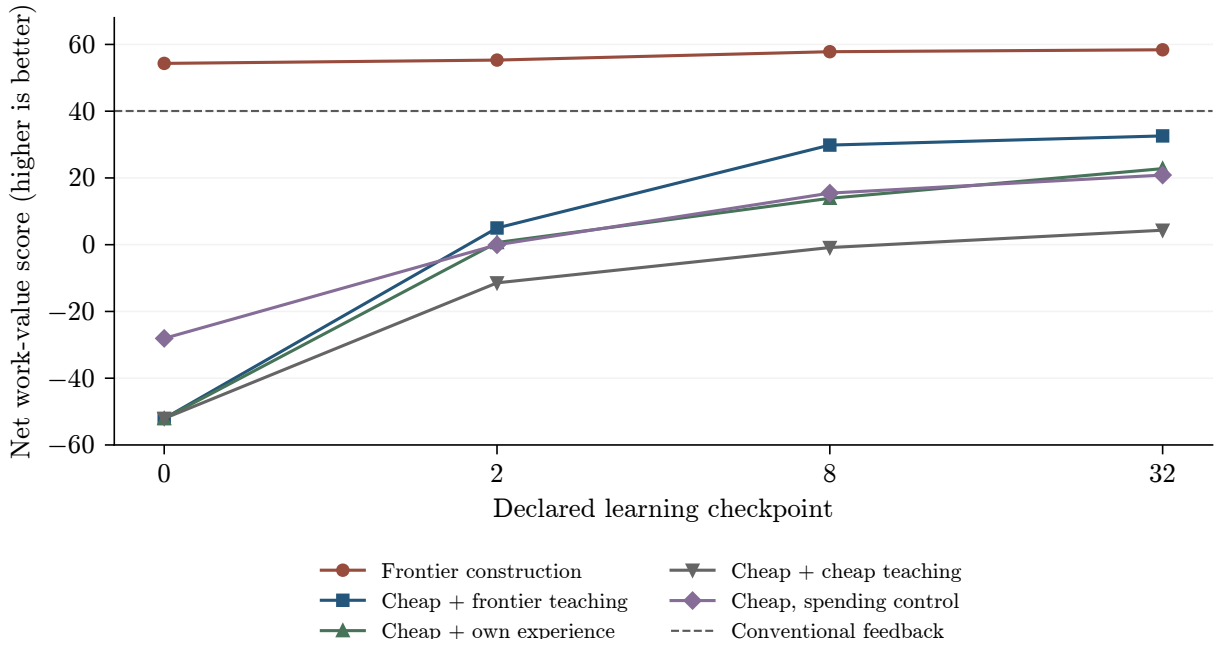


Figure 2: Whole-system learning in twelve unknown-mechanism businesses

Notes: Descriptive means across twelve independent businesses, each with two learning repeats. Outcomes use four protected scenarios per artifact and repeat. Simultaneous uncertainty is broad: the initial frontier-minus-cheap difference is 106.39 with interval $[-160.36, 242.30]$; the selected endpoint minus contemporary frontier is -25.80 with interval $[-218.88, 193.08]$. These main data do not replace the paused fresh confirmation.

The observed quality hierarchy is accompanied by a construction-cost hierarchy. Initial cheap and frontier acquisition cost about \$0.0043 and \$1.1252 per system. At checkpoint 32, frontier-taught cheap acquisition is \$2.0447–\$2.3504 and frontier acquisition \$5.6210–\$6.1451, retaining unresolved charge allowances. Their measured operating costs are tiny: approximately \$0.000050 and \$0.000068 per episode including a CPU proxy. In contrast, repeated initial frontier inference costs about \$5.22–\$5.26 per episode. All 96 frontier checkpoint records are zero-call artifacts. Frontier construction is therefore itself a cheap-operation alternative, not necessarily a source of recurring frontier demand.

These developmental results are suggestive but imprecise. The twelve-business intervals are too broad to establish the key performance comparisons. Initial spending controls achieve their prescribed acquisition match, whereas later matching limitations remain disclosed; later teaching gains cannot be called a clean equal-spending effect. At the user’s direction, the original confirmation was paused incomplete and resources were redirected to a small handoff experiment. Its unfinished outcomes are not incorporated as confirmatory evidence here.

7 Frozen Handoff and Fresh Software Validation

7.1 A handoff that eliminates the operating model

The smaller handoff study follows two businesses, one with groups of three units and one with a group of six, with two independent development repeats each. The initial system asks Astra for daily actions. The saved route is `openai/gpt-6-astra`, medium reasoning, with a 4,096-token output allowance and at most two requests per day including a repair of an invalid answer. Its scaffold

retains eight recent observations and actions and a bounded notes field.

At checkpoints 0, 2, and 8, frontier development can preserve code, prompts, memory, procedures, retrieval, and tools. The planned primary comparison changes the operating model while holding that state fixed. Development uses 32 collection episodes and 28 additional development-evaluation episodes across the four lineages; the label “eight” therefore does not mean eight total exposures per artifact.

Three lineages produce zero-call software at checkpoint two and the fourth at checkpoint eight. The predeclared all-zero-call branch stops the handoff before fresh performance evaluation. This is an architectural result, not evidence that a cheap operating model preserves quality: there is no operating model left to replace. It also does not establish an initial frontier advantage. A separate prospective extension subsequently evaluates all four unchanged final artifacts on fresh scenarios.

7.2 Design of the final validation

The validation contains 24 independent paired batches. Each includes both fixed businesses, both software repeats for each business, the common initial-frontier reference, and a conventional feedback controller. There are 96 software, 48 frontier, and 48 conventional episodes, totaling 192. The reference is evaluated once per business and batch and shared across software-repeat comparisons. Each batch’s primary observation averages the software repeats within business, subtracts that business’s frontier score, and averages the two business differences equally. The independent sample size is 24, not 192. Memory resets between episodes, external shocks are paired, and actions generate each controller’s own trajectory.

The criterion is an expected software-minus-frontier difference greater than -2 score units. A predeclared bounded-mean betting interval must have its 95% lower endpoint above that threshold. Its support is justified analytically, including possible severe failures, rather than estimated from the largest observed error. Inference conditions on these businesses and programs; it does not estimate average performance over all possible businesses.

Table 5: Fresh business performance and recurring model use

Business	Controller	Episodes	Mean score	Calls/episode
A	Conventional feedback	24	44.977	0
A	Initial frontier operator	24	50.891	60.21
A	Software, repeat 1	24	55.152	0
A	Software, repeat 2	24	58.626	0
B	Conventional feedback	24	53.781	0
B	Initial frontier operator	24	62.210	60.21
B	Software, repeat 1	24	70.465	0
B	Software, repeat 2	24	71.074	0
Both	Software, pooled	96	63.829	0
Both	Initial frontier, pooled	48	56.551	60.21
Both	Feedback, pooled	48	49.379	0

Notes: Each episode is a simulated sixty-day business run. A has resource groups of three; B has a group of six. Higher scores indicate greater net completed-work value, not dollars. All 192 episodes complete without missing outcomes or controller failures. The single primary inference is the paired batch-level comparison; individual rows are descriptive.

7.3 What the fresh outcomes establish

All four programs have higher sample means than their frontier references (Table 5). Pooled mean software performance is 63.829, compared with 56.551 for frontier operation and 49.379 for feedback. The mean paired gain is 7.278. Every one of the 24 batch gains is positive, ranging from 5.75 to 11.12. This consistency is descriptive; it does not replace the predeclared test of mean performance.

The software makes no model calls in any fresh episode. Frontier operation averages 60.21 calls and \$4.96 in API expenditure per sixty-day run. Software execution averages about 3.16 measured CPU seconds, with a conservative allowance of 360 seconds including startup and residual work. At an illustrative \$0.05 per CPU-hour, that is approximately \$0.000044–\$0.005 per run. Serialized artifacts occupy roughly 20–29 KB each, plus a shared 43.5 MB execution image. Zero API use does not mean zero computation, engineering, or energy consumption.

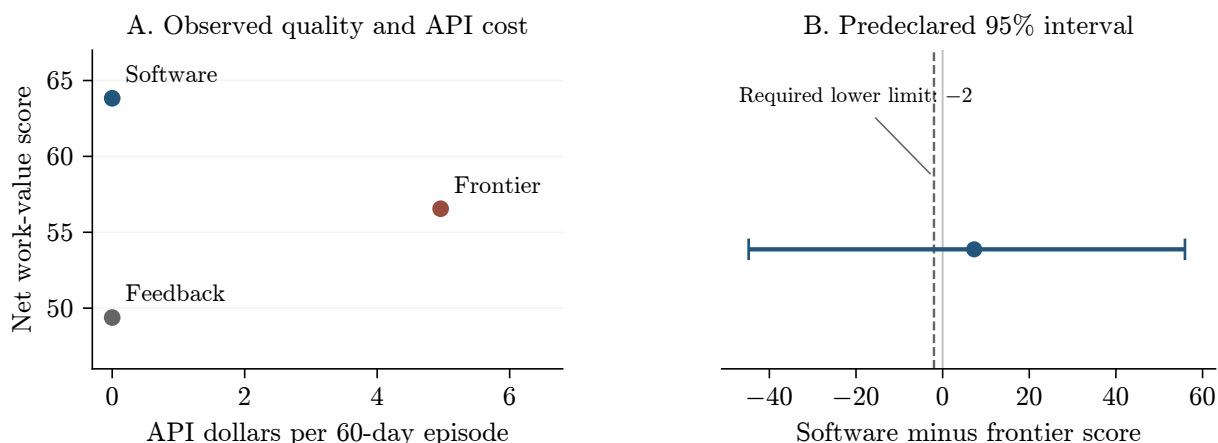


Figure 3: Fresh software performance, operating expense, and statistical uncertainty

Notes: Left: sample means, excluding acquisition and local execution costs from the horizontal axis. Right: the unchanged primary 95% confidence interval. Its lower endpoint must exceed -2 to establish quality preservation. This condition is not met.

The formal preservation test remains inconclusive: the predeclared 95% interval is $[-44.778, 55.938]$. The lower endpoint is far below -2 . The favorable observed sample and the conservative population statement are different results, and both must be reported.

This precision limitation was known before execution. The phase proceeded as a diagnostic fallback after a stringent certificate was forecast to exceed its budget. Under a prospective hypothetical constant gain of six units, the chosen method required 174 batches and about \$1,879 in new API charges; this was a sensitivity calculation, not a power guarantee. The executed 24-batch design was not well matched to resolving the declared tolerance. Possible severe failures make the analytic support broad even though none occurred, and the chosen interval does not convert the tightly clustered observed gains into a sufficiently tight certificate. A different analysis or further evaluation would require its own justification; no replacement interval is introduced in this synthesis.

7.4 Acquisition and conditional repayment

Fresh validation costs \$237.99 in settled API charges across 2,890 requests. Adding the complete \$119.87 handoff-development bill once gives \$357.86. With historical and validation CPU priced at the stated proxy, acquisition is approximately \$357.87–\$358.95. This is the four-program handoff-plus-validation account, not total research-program expenditure.

Charging that whole bill to a portfolio deployed uniformly across its four systems implies repayment after approximately 73 further deployments at observed recurring costs. Adding \$100, \$500, or \$1,000 in engineering raises conservative rounded repayment counts to 93, 174, and 275. These are conditional accounting calculations, not certified profitable deployments. They assume acceptable quality, stable savings, and continued usefulness, while omitting unpriced maintenance, hosting, storage, and the value of decision losses. A deployment is a sixty-day simulated business episode, not an observed customer transaction.

8 Synthesis and Economic Interpretation

The experiments establish several different facts that should not be compressed into one claim. In disclosed small problems, inexpensive models often construct useful software immediately. When the relevant mechanism must be identified within a supplied class, experience and further development improve performance without a demonstrated material advantage from frontier teaching. In richer business settings, frontier construction has a substantial observed advantage, yet the resulting controllers can operate without further frontier inference. Finally, fresh scenarios show that four such artifacts continue to function and have favorable average performance; the formal quality certificate remains unresolved.

The evidence is strongest for *reusable control* and weaker for *experience-driven substitution of intelligence*. The final comparison changes both experience and architecture: the initial system makes model calls, whereas the final system has been allowed to develop software. There is no matched zero-experience construction control in that handoff phase establishing that the same program could not have been built immediately. The four-family experiment offers stronger evidence of useful learning but weaker initial need for frontier capability. Neither fills the other's identification gap.

The appropriate economic interpretation is that model-assisted development can create an asset. Inference demand depends partly on whether customers rent decisions or acquire procedures, how often procedures can be reused, and how quickly specifications change. The repair pilot demonstrates the importance of that last condition. The contrast between near-zero API operating expense and potentially substantial acquisition costs also cautions against treating deployment savings as a sufficient welfare or profitability result.

Conventional control remains central. Exact planning solves every tested configuration in the structural inventory study, and feedback rules provide competitive business baselines. A frontier advantage over a particular cheap model is not evidence that every inexpensive algorithm would fail. Conversely, a conventional rule's success in these environments does not establish that engineering a comparable rule is always inexpensive.

Many AI applications can be described as decisions under uncertainty once their objectives, information, and consequences are specified. This supplies a common language, not a universal result about cheap controllers. A claim that simple controller descriptions are dense in a broader space would require a distance or topology, an approximation theorem, and a link from approximation to performance and computational cost. None is supplied by these experiments. Compact descriptions,

learnability, and cheap execution remain distinct properties. The present evidence supports the tested task distributions and fixed business panel; market prevalence, energy savings, and provider margins require other evidence.

The resulting conclusion is substantive but bounded. Repeated uncertainty does not, by itself, force repeated purchases of language-model inference: useful stochastic decisions were repeatedly carried out by reusable programs. The research program has not established the stronger transition from demonstrated frontier necessity to inexpensive operation caused by accumulated experience. It has identified construction, information acquisition, reuse, and maintenance as separate mechanisms through which that transition could occur—and the comparisons needed to distinguish them.

References

- Kenneth J. Arrow. Economic welfare and the allocation of resources for invention. In *The Rate and Direction of Inventive Activity: Economic and Social Factors*, pages 609–626. Princeton University Press, 1962a. URL <https://www.nber.org/books-and-chapters/rate-and-direction-inventive-activity-economic-and-social-factors/economic-welfare-and-allocation-resources-invention>.
- Kenneth J. Arrow. The economic implications of learning by doing. *The Review of Economic Studies*, 29(3): 155–173, 1962b. doi: 10.2307/2295952. URL <https://doi.org/10.2307/2295952>.
- Erik Brynjolfsson, Daniel Rock, and Chad Syverson. The productivity J-curve: How intangibles complement general purpose technologies. *American Economic Journal: Macroeconomics*, 13(1):333–372, 2021. doi: 10.1257/mac.20180386. URL <https://www.aeaweb.org/articles?id=10.1257/mac.20180386>.
- Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning. arXiv:1803.07055, 2018. URL <https://arxiv.org/abs/1803.07055>.
- Benjamin Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2:253–279, 2019. doi: 10.1146/annurev-control-053018-023825. URL <https://arxiv.org/abs/1806.09460>.
- Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. arXiv:1511.06295, 2015. URL <https://arxiv.org/abs/1511.06295>.
- Ian Waudby-Smith and Aaditya Ramdas. Estimating means of bounded random variables by betting. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 86(1):1–27, 2024. doi: 10.1093/jrssb/qka d009. URL <https://arxiv.org/abs/2010.09686>.

A Evidence Register and Completion Status

This paper is a retrospective synthesis of completed saved evidence as of October 12, 2026. It introduces no experimental observations, model calls, revised selections, or replacement confidence claims. The source bundle includes the protocols and reports listed below, a hash manifest, saved numerical aggregates, and a script that rebuilds the principal figures and numerical entries. Full simulator and billing replay require the preserved repository. Paused evaluation data were not consumed for this paper.

Table 6: All experimental stages and their evidentiary status

Stage	Completed scope	Result or remaining limitation
Direct inventory	72 paid cells; 1,728 episodes; 192 baseline episodes.	Frontier usually close to exact reference; effort and text-learning gains uneven.
Deeper text learning	Two lineages; fourteen selection candidates; 2,000 fresh episodes per condition.	Neither condition earns the one-unit certificate.
Executable construction	Three tiers; one lineage each; 32 episodes each.	All zero measured exact regret; cheap construction already works.
Frozen transfer	Three unchanged programs; 64 cases each.	All zero measured exact regret at larger horizon and capacity.
Structural prediction	Eighteen configurations; 81 solver cells; 48 held-out cases.	Full planning always passes; router offers no aggregate advantage.
Reuse and repair	Three families; two builds per tier/family; 864 reusable evaluations.	Exact results; contract changes break unrepaired software; repair restores it.
Finite-model learning	Pilot, sixteen-task main, eight-task structural holdout, 1,024-task confirmation.	Learning useful; frontier teaching equivalent at declared tolerance; selected near-optimality inconclusive.
Vending adaptation	Eighteen conventional ninety-day rollouts.	AI comparison and fresh confirmation not completed; not a model-effect result.
Business screen	Six businesses; two builds per tier/business.	Apparent frontier advantage, uneven competence; later checkpoints not run in this phase.
Intermediate curriculum	Three businesses; two repeats; five completed core curves.	Affordable curves deteriorate; remaining controls and confirmation superseded before completion.
Information disclosure	Two businesses; two repeats; three information conditions; two tiers.	Disclosing rules does not eliminate observed construction gap.
Whole-system main	Twelve businesses; two repeats; five arms; four checkpoints.	Frontier construction leads observed means; original confirmation paused incomplete.
Small frozen handoff	Two businesses; two repeats; checkpoints 0/2/8.	All endpoints zero-call; smaller-model comparison stops before fresh evaluation.
Fresh software validation	Same four programs; 24 fresh paired batches; 192 episodes.	Favorable software means and zero calls; formal preservation inconclusive.

Repository sources. Inventory and its extension: `docs/report.md`, `docs/protocol.md`, and `docs/extensions/PROTOCOL.md`. Executable construction, transfer, structure, and reuse: `compression_reporting/REPORT.md`, `transfer_h5_reporting/REPORT.md`, `structure_reporting/REPORT.md`, and `reuse_reporting/REPORT.md`. Finite-model phases: `learning_reporting/STUDY_REPORT.md`, with its pilot, main, holdout, and confirmation reports. Vending status: `vending_reporting/COMPLETION_AUDIT.md`.

Business sources are `rules_reporting/FINAL_REPORT.md`, `rules_learning/MAIN_PROTOCOL.md`, the completed core curves under `rules_learning_artifacts/main_core_reports/`, the information diagnostic under `rules_learning_artifacts/information_reports/`, and the main report under `rules_agent_artifacts/reports/a78709eda86fb4a68f771f2213f6418b749613aada4e8c78f91ff773855fe0bf/`. Handoff

evidence is `rules_handoff_short_artifacts/final/20261010T2135/REPORT.md`. Final validation uses `final_software_validation/PROTOCOL.md`, `FORECAST.json`, `REPORT.md`, and the analysis digest beginning `38ef35b85aa56b34`. Exact source paths and content hashes are in the bundled `source_manifest.json`.

B Inference, Accounting, and Protocol Qualifications

Inventory precision. The original small-cell analysis uses a frozen 900-unit support bound with multiplicity adjustment. At 24 episodes its Hoeffding penalty is approximately 315 units, far too wide for a one-unit tolerance. The later confirmation uses analytically certified bounds 171.07 and 106.72, a variance-sensitive betting mixture, error probability 0.025 per condition, looks every fifty cases, and a cap of 2,000 per condition. The shared-condition final-look upper limit is 1.5570; 1.5228 is the valid running minimum of its confidence sequence. Neither replaces the original small-cell analysis retroactively.

Finite-model confirmation. Five claims receive error probability 0.01 each, giving at least 95% simultaneous coverage over statements and looks. The declared raw support is 24 and normalized support one. A separately frozen pre-evaluation precision amendment extends the maximum from 768 to 1,024 tasks while preserving the candidate, method, and earlier looks. The partial-observation family has selected mean regret 0.2654, compared with 0.0583 contextual, 0.0794 sequential, and 0.0710 compositional; these are descriptive subgroups, not separate confirmatory conclusions. Operational affordability requires valid source and memory, measured CPU within one second, no deployment API, and learning API expense within one dollar, along with the protocol’s storage limits.

Final business interval. The primary method inverts a mixture of sixteen bounded-mean betting strategies, with fractions 2^{-j} for $j = 0, \dots, 15$ and threshold 40 in each direction, following the general bounded-mean approach of Waudby-Smith and Ramdas (2024). Outward rounding preserves numerical coverage. The analytically justified paired support is approximately $[-215.7773, 215.7773]$, including possible failures. All 24 scheduled batches enter. The interval addresses the fixed panel and evaluation-time model route. Observed extrema, normal approximations selected after seeing the data, and thousands of within-episode decisions are not substituted for the frozen analysis. The design’s low precision limits the claim despite favorable fresh means.

Incomplete work. The vending adaptation lacks completed paid AI comparisons. The intermediate software curriculum’s incomplete spending control, planned confirmation, and structural holdout are not negative results. The whole-system main’s original confirmation remains paused and incomplete. The small handoff ends according to its zero-call branch; fresh software validation is a distinct prospective extension and does not silently complete the smaller-model comparison. No additional software-only precision refinement has been executed for this paper.

Costs and liabilities. The first seven studies through October 5 record \$525.93 in provider-reported API charges and \$31.52 in unresolved-charge allowances, for a planning total of \$557.46. Those allowances cover 86 attempts and are not verified charges or guaranteed billing caps. The initial business screen separately records \$7.48. The final handoff and validation account records \$357.86, fully settled. The intervening curriculum and whole-system records have their own expenditure and liability accounts. We do not manufacture a program-wide reconciled total by adding overlapping standalone recipe costs or treating unresolved calls as free. Historical audit and transport qualifications remain in the preserved reports.

The costing exercises disclose unpriced human engineering, coding assistance outside study ledgers, and some host, storage, and maintenance costs. API expenditure is neither measured energy use nor the economic value of completed work. The full repository retains failures, retries, raw evidence, and prospective execution amendments. This manuscript preserves those distinctions without reproducing operational audit logs in the main text.