COGNITIVE

WHITE PAPER / AI LITERACY & CONSULTING

Compiling Capability

From powerful models to reliable systems: enterprise-owned intelligence through model-independent harnesses.

David C Robinson

October 2026

Complete text of the supplied paper, reformatted for the Cognitive SLC edition. External citations and historical dates are preserved. Performance estimates, forecasts, and corporate examples are author-supplied analysis, not independently verified customer results.

EXECUTIVE PERSPECTIVE

Stop buying the same reasoning twice

Some capability purchased from a powerful model on every run can instead be engineered into a reusable operating procedure. The business opportunity is to know which capability, for which work, and at what total cost.

A recurring business task contains more than a reasoning problem. It contains an operating procedure: which evidence to gather, which rules apply, what must be checked, when to retry, and when to stop. When a model is asked to rediscover that procedure on every case, the organization pays repeatedly for decisions that may already be understood.

This paper advances a different approach. Use human expertise and, where useful, a stronger model to construct and test the procedure. Capture the repeatable parts in executable structure. Then use the minimum sufficient runtime model for the remaining cognitive work. We call this **compiling capability**: distillation into structure rather than into model weights.

The evidence is meaningful but bounded. DSPy, Salesforce, Stripe, and Intercom document complementary mechanisms for making prompts, procedures, checks, and production learning explicit. AlphaEvolve demonstrates that model-assisted discovery can produce reusable code. None establishes universal equivalence between model classes.[1–6]

Recent industry discussion has converged on ownership. Sequoia Capital describes the application-layer contest as a fight for the intelligence layer and asks, “who should own the intelligence at the core of your business?” Its technical playbook emphasizes evaluations, harness and context engineering, post-training, and learning from production. This paper addresses the complementary enterprise question: how are those assets controlled, verified, and released?[14,15]

The three arguments fit together without collapsing into one claim. Sequoia explains why the enterprise should own the intelligence-producing system. Lectotype explains how that system can govern context, procedures, evaluations, trajectories, and release authority

while remaining portable across models. Research from MIT Sloan and its coauthors explains why reorganizing contiguous work around that system can create economic value by reducing coordination costs and changing the boundaries of tasks and jobs.

[14,15,18]

Four implications for enterprise leaders

Change the unit of procurement. Compare complete model–harness systems against a business acceptance standard. Token price and model benchmarks do not establish the cost of an acceptable work product.

Own the intelligence-producing system. Retain the context, procedures, evaluations, trajectories, and release authority that make models useful. Model access alone does not create enterprise ownership.

Treat procedures as assets. Version the workflow, its authoritative evidence, its checks, and its validity basis. Reuse creates value only while those artifacts remain correct and useful.

Make verification the investment gate. An attractive automation rate is not enough. Measure incorrect outputs that are accepted, unfamiliar cases that are missed, and the work humans must still perform.

01 / THE STRATEGIC SHIFT

The workflow, not the model, is the economic unit

Before upgrading the model, identify the work you are asking the model to do.

A model-centric agent receives an objective, chooses tools, interprets results, and decides what to do next. That flexibility can be valuable when the path is genuinely unknown. But a repeatable operation does not necessarily require the same freedom. If the required source, calculation, or approval is already known, asking the model to choose it again adds a decision rather than removing one.

A **cybernetic harness** places the model inside an explicit feedback system. The controller manages state, gathers required evidence, invokes bounded cognitive work, measures the result, and selects correction, completion, or escalation. Tools are embedded in these execution patterns. They are not simply presented as a larger menu for the model to navigate.

EXHIBIT 1

Move repeatable executive decisions into the operating procedure

Responsibility	Model-centric agent	Control-centric harness
Evidence acquisition	Model decides what to find	Policy specifies sources and prerequisites
Tool execution	Model selects and sequences calls	Controller runs approved patterns
State and recovery	Model interprets conversational history	System tracks state and bounded recovery
Completion	Model decides the answer is finished	Acceptance checks permit completion or escalation

Source: author’s architecture. These are design poles; real implementations can mix them.

The distinction is not “thinking versus no thinking.” The model may still interpret an ambiguous passage or reconcile competing explanations. The change is that it no longer has to manage every part of the global process while doing that local work.

This suggests a practical diagnosis. A failure caused by missing evidence calls for better evidence acquisition. A forgotten validation step calls for enforced validation. A genuinely difficult interpretation may require a stronger model or expert. Treating all three as a single model-quality problem obscures the available remedies.

Our proposition is that some observed performance differences between model classes reflect this global executive burden. How much of the difference can be recovered is an empirical question. The test is a completed workflow at an agreed quality, latency, and risk level – not whether the smaller model sounds as fluent as the larger one.

The economic unit is a safe chain, not an isolated task

Demirer and colleagues model production as an ordered sequence of interdependent steps. Their central result is that automation depends not only on which steps AI can perform, but also on where those steps sit relative to one another. Adjacent AI-suitable steps can form a continuous chain; the same steps

scattered across a workflow create repeated handoffs and coordination costs. Their empirical evidence is consistent with AI-executed steps clustering in chains.[18]

This extends the architecture thesis. A harness does more than improve an individual call. It changes the boundary of the work by carrying evidence, state, calculations, checks, and recovery across several adjacent steps. The relevant comparison is therefore not model versus person at each micro-step. It is the cost and reliability of the complete chain under a defined acceptance standard.

Compile the longest safe chain that can meet the acceptance standard inside an approved authority boundary.

Longest safe is not longest possible. Deterministic checks can run inside the chain without becoming human handoffs. People remain at boundaries where judgment, consequence, or authority requires them: unresolved evidence, material exceptions, external commitments, irreversible actions, or changes to the governing policy.

Human-in-the-loop is not a single design. Requiring a person to inspect every intermediate output may add reassurance, but it can also recreate the coordination cost the system was intended to remove. A governed chain should complete low-discretion, checkable work without interruption and surface the evidence needed at the next meaningful decision.

The architecture should distinguish internal verification from transfer of responsibility. Schema validation, source reconciliation, policy checks, and bounded repair can remain inside the controller. Human review belongs where the system cannot adequately detect a consequential error or where only a named person may authorize the result. This is risk-bounded chaining, not autonomy for its own sake.

02 / COMPILING CAPABILITY

Spend intelligence on discovery; reuse the result

The reusable artifact is not a transcript. It is a procedure the organization can inspect, execute, test, and change.

Compilation begins by separating what varies from what repeats. A supplier’s facts vary; the evidence required to establish those facts may not. A support request varies; the conditions for a refund may be stable. A software defect varies; the requirement to run relevant tests should not depend on whether a model remembers it.

A stronger model can help propose task decomposition, source requirements, checks, and exception routes. A domain owner supplies or approves the basis for those choices. The resulting artifact should contain executable transitions, bounded model instructions, tool bindings, acceptance checks, and stopping rules. A prose description alone does not transfer control.

EXHIBIT 2

Make the procedure reusable before reducing the runtime burden

Control structure

A — Expert + stronger model Design and evaluate
B — Released artifact Policy, checks, evidence
C — Bounded worker Local cognitive work
D — Verification Accept or escalate

A → B; B → C; C → D

Discover once; validate before release. Reuse the structure on each covered case.

Source: author’s architecture. The arrows represent control and release relationships, not measured performance. The design combines documented mechanisms and is not presented as a measured deployment of any cited company or of Lectotype.

This is distinct from routing and from weight-based distillation. Routing chooses a model for an incoming task. Fine-tuning or distillation changes a model. Compiling capability changes the surrounding program and may reduce the capability required of each runtime call. These approaches can be combined, but their costs and effects should be measured separately.

The phrase **distillation into structure** is an architectural analogy, not a claim that neural representations have literally been translated into code. DSPy supplies a related precedent for optimizing composable language-model programs; its Assertions work adds programmed constraints and refinement. Workflow discovery, prompt optimization, and weight optimization remain different interventions.[1,2]

Ownership extends beyond the model

The emerging ownership argument is often described as a choice between renting a frontier model and operating or post-training an open-weight model. That choice matters, but it is incomplete. An enterprise can host model weights and still depend on someone else for the context assembly, workflow logic, evaluation criteria, production traces, or release process that determines how the model performs useful work.

Sequoia’s sovereign-AI framework organizes the stack around the model, context, and harness. Harrison Chase describes the harness as the component that brings the right context to the model at the right time

and makes the system inspectable. In this paper’s terms, the harness also carries the approved procedure, permissions, checks, recovery behavior, and stopping conditions.[14,15]

Enterprise ownership therefore means preserving the intelligence-producing system around the model. Business policies, evidence authority, acceptance tests, exception histories, and release decisions must remain versioned assets the organization can inspect and move. The model can then be rented, hosted, tuned, or replaced without surrendering what the organization has learned about performing the work.

Model classes are functional shorthand

We use small/fast, balanced, and flagship as operational categories. Haiku-Sonnet-Opus and OpenAI's Luna-Terra-Sol are illustrative labels, not claims of equal benchmark performance, parameter ratios, or interchangeable behavior across vendors. OpenAI describes the latter family in those broad product roles.[7] Each deployed model-harness pair still requires evaluation.

Compilation can be useful without a larger-model author. The stronger-model-assisted version is a separate hypothesis about how cheaply and effectively organizations can build the reusable structure.

03 / A PUBLISHED BENCHMARK AND OUR TEST TARGET

AFlow shows the kind of evidence Lectotype must produce

These are published third-party results, not Lectotype results. They are included as a temporary benchmark for designing our own controlled evaluation.

AFlow is a research framework that uses a stronger model to search over code-defined workflows and execution feedback. In the authors' HumanEval experiment, GPT-4o-mini executing an optimized workflow scored 94.70%, compared with 93.89% for direct GPT-4o invocation. The reported execution cost was 8.05% of the direct GPT-4o comparator. Lectotype has not run or reproduced this experiment.

EXHIBIT 3

The published comparison sets the form of the test

Execution model	Direct invocation	AFlow workflow[1]
GPT-4o-mini	87.02%	94.70%
GPT-4o	93.89%	96.20%

Source: Zhang, J., et al., “AFlow: Automating Agentic Workflow Generation,” ICLR 2025; arXiv:2410.10762v4, Section 5.1 and Appendix D.[1] Workflow optimized for GPT-4o-mini. Scores are the paper’s reported HumanEval test-split results. The 8.05% ratio compares execution costs and excludes workflow-discovery costs.

The evidence is real and relevant, but bounded. The larger model also improved when given the transferred workflow, from 93.89% to 96.20%. The result therefore shows that a less expensive model-plus-workflow can exceed a larger direct-invocation baseline on defined work. It does not show that the smaller system dominates the best larger-model configuration, establish statistical equivalence, or measure enterprise deployment and lifecycle costs.

This is the architecture-level proposition Lectotype must now test with its own governed workflows.

The benchmark Lectotype should run

Use one bounded workflow, objective acceptance criteria, and an independent holdout set. Compare four configurations on identical cases: larger and less expensive models invoked directly, then each inside the same Lectotype harness. Hold the data, tools, permissions, and review standard constant to separate model effects from harness effects.

Report task success with confidence intervals; unsupported claims and critical errors; evidence accuracy; rule compliance; trace completeness; escalation and review time; latency; and cost. Separate workflow discovery, setup, and maintenance from runtime cost. Preregister the study and blind human reviewers to the system configuration.

Our target is the AFlow pattern: match or beat direct larger-model quality on defined work, exceed it on governance and traceability, and materially reduce cost. The 8.05% ratio is a stretch reference, not a Lectotype result.

04 / THE PATTERN IN BUSINESS SOFTWARE

Companies are making control structure explicit

Production accounts document the mechanisms. Their existence should not be confused with a controlled model-substitution result.

Stripe: the workflow can insist on a check

Stripe’s Minions engineering account describes code-defined blueprints, including a deterministic lint step inside the development loop. Stripe also reports more than a thousand merged pull requests weekly, with human review. That volume indicates use, not the success rate or causal productivity gain. The architectural lesson is the placement of the check: it runs because the process requires it, not because the model elects to run it.[3]

Salesforce: behavior becomes an execution artifact

Salesforce describes Agent Script as a declarative language for conditional logic, actions, rules, and handoffs. It compiles into an Agent Graph consumed by the Atlas Reasoning Engine. This makes the control structure explicit. It demonstrates compilation of a supplied specification, not automatic discovery of the right specification or proof of a cheaper runtime model.[4]

Intercom: procedures have a lifecycle

Intercom’s Fin Procedures combine natural-language instructions with deterministic controls, data connections, conditions, code, and handoffs. Its simulation library can be rerun when products, policies, or procedures change; Raylo and MONY Group are named production examples. This supports a repeatable lifecycle: define, test, deploy, and revalidate. It does not establish that fewer handoffs necessarily mean fewer errors.[5]

EXHIBIT 4

Three implementation patterns transfer different responsibilities

Pattern	Reusable artifact	Management question
Enforced execution	Approved steps and checks	Which actions must occur regardless of model choice?
Compiled behavior	Specification and execution graph	Who owns the rules and approves changes?
Procedure lifecycle	Procedure plus regression tests	What change invalidates the previous approval?

Sources: Stripe, Salesforce, Intercom.[3–5] Management questions are the author’s interpretation.

Together, these cases show practical ways to move process responsibility out of free-form reasoning. The white paper’s proposal is to connect those mechanisms to a disciplined model-selection decision: what runtime capability remains necessary after the stable procedure has been made explicit?

05 / REUSE AND RESTRAINT

A harness is valuable only relative to the alternatives

The same evidence that supports compilation also argues against adding orchestration indiscriminately.

AlphaEvolve: sometimes the runtime model disappears

Google DeepMind’s AlphaEvolve combines language models, automated evaluation, and evolutionary search to discover programs. Google reports that a resulting Borg scheduling heuristic recovered an average of 0.7% of Google’s worldwide compute resources. The artifact is human-readable code.[6]

This illustrates an endpoint of the argument: expensive discovery can produce a program that no longer needs a model for each execution. For a business architect, deterministic code must remain a candidate alongside smaller models, larger models, and hybrid systems. “Agentic” is not a requirement for capturing the value of model-assisted discovery.

Anthropic: improvements can make scaffolding obsolete

Anthropic describes removing explicit context resets when moving from Sonnet 4.5 to Opus 4.5, then removing a sprint structure with Opus 4.6. Its account also discusses the continuing value of evaluation near difficult task boundaries.[8] These are engineering observations, not a controlled universal comparison of product tiers.

The implication is to evaluate the pair. Retire reasoning support that no longer improves the outcome. Retain permissions, evidence requirements, and approval controls because their purpose is different. A model becoming better at following a rule is not the same as the organization no longer needing to enforce that rule.

Reported outcomes are leads, not portable promises

In OpenAI’s Agents API announcement, SafetyKit reports a 60% reduction in cost per case while maintaining existing performance. Hypha reports 86% fewer failed agent responses after separating the harness from the sandbox. These are attributed customer statements. The publication does not disclose enough methodology to isolate model, infrastructure, task-mix, or evaluation changes.[9]

Do not infer an enterprise savings rate from someone else’s architecture story. Use the story to identify a mechanism; use your own workload to establish its value.

The evidence supports a design agenda, not a universal savings forecast. That distinction permits a strong conclusion without overstating it: reusable structure is a credible source of capability, and its contribution must be measured against today’s best practical alternative.

06 / THE REFERENCE ARCHITECTURE

Use one loop to perform the work and another to improve it

A durable system separates execution, learning, and release authority.

The proposed architecture has two connected loops. The inner loop executes an approved policy. The outer loop examines exceptions and audited outcomes, proposes changes, tests them, and releases a new version. The distinction keeps learning from becoming unreviewed production behavior.

EXHIBIT 5

The runtime executes approved policy; a separate process changes it

OUTER LOOP / Improve the procedure

Control structure

A — Review exceptions and audited outcomes
B — Propose a change using expertise + models
C — Test independently against fixed standards
D — Authorize a version with rollback
E — Gather Approved evidence
F — Act Code + bounded cognition
G — Verify Acceptance checks
H — Route Complete, repair, escalate

B → C; C → D; F → G; G → H; D → A; H → E

Source: author’s architecture. The arrows represent control and release relationships, not measured performance.

In the inner loop, a Gather–Act–Verify pattern makes control concrete. **Gather** obtains the required evidence through approved adapters. **Act** combines deterministic operations with bounded model work. **Verify** checks the result against its acceptance contract. The controller then completes the task, requests a specific repair, or escalates. A general request to “try harder” is not an error-handling policy.

The outer loop turns a recurring exception into a candidate improvement rather than immediately teaching the runtime to accept it. A change might add a document requirement, a branch, a test, or an explicit exclusion. A stronger model can assist in drafting that change; it should not authorize its own release.

Four artifacts make the design inspectable: a **task policy** specifying permitted behavior; **evidence records** linking claims to sources; **check results** recording what was evaluated; and a **run record** preserving versions, transitions, interventions, and costs. These are operational records, not a promise to expose a model’s private internal reasoning.

The acceptance boundary must remain separate. A candidate procedure must not weaken its own tests, change authoritative records, grant itself permissions, or promote itself to production. Without that separation, optimization can improve the reported metric by changing what counts as success.

07 / EMBEDDED CONTROL PATTERNS

Autonomy is a design choice, not a maturity contest

More agents, more tools, and more loops do not automatically produce a better system.

Control patterns can be embedded at different levels. A field extractor can run inside a local validation-and-repair loop. Several validated extractions can feed a reconciliation step. A larger workflow can require every subtask to return a typed result and an evidence record before composing the final answer. Each pattern needs its own budget and stopping rule; the enclosing controller remains accountable for the final outcome.

This is the distinction between assembling capabilities and presenting choices. The extraction worker does not need to understand all source adapters, approval paths, or downstream agents. It needs the context for one operation and a contract for its result. Complexity has not disappeared; it has moved into an explicit, testable program.

EXHIBIT 6

The H4–H5 boundary is whether the control policy itself changes

Level	Control arrangement
H0 / Wrapper	Prompt and output formatting; the model handles the task.
H1 / Toolbelt	The model largely plans and chooses tools.
H2 / Scaffold	Templates, memory, and retries share the burden.
H3 / Orchestrator	The system selects stages, sources, tools, and model calls.

H4 / Cybernetic	An approved policy measures outcomes and controls correction, stopping, or escalation.
H5 / Adaptive cognitive architecture	The system constructs or revises its control policy during operation, subject to separate acceptance.

Source: author’s proposed taxonomy. These are architectural categories, not an established industry maturity scale.

An H4 system can choose different branches dynamically while keeping the policy fixed. Offline model-assisted compilation can improve that policy without making the runtime H5. Conversely, H5 concerns changing the procedure itself, not merely generating another answer or assigning another subagent.

For a repeatable business process, a bounded H4 design may be preferable to H5. It can preserve useful adaptability while making release authority and failure behavior easier to specify. The objective is the least complex architecture that meets the operating requirement, not the highest label.

08 / WHERE TO START

Select the workflow before selecting the model tier

The best candidates combine reusable decisions, observable outcomes, and a manageable rate of change.

A task’s suitability is not determined by how impressive it sounds. A seemingly modest request may require difficult judgment because the source evidence is missing or contradictory. A complex-looking operation may be tractable because its steps and acceptance tests are explicit. The relevant question is what remains uncertain after the system has gathered the right evidence and applied the known procedure.

EXHIBIT 7

Four questions identify the practical substitution opportunity

Question	Favorable condition	Design response when weak
Can the work be decomposed?	Bounded steps preserve the relationships needed for the answer.	Keep synthesis with a stronger model or expert.
Can the result be checked?	Checks detect substantive errors, not only formatting.	Use assisted work with explicit review.
Can exceptions be recognized?	Known exclusions and uncertainty have defined routes.	Restrict scope and measure missed exceptions.
Will reuse repay change?	Enough repetitions occur before the procedure needs substantial revision.	Prefer a simpler workflow or direct model use.

Source: author’s decision framework. It is not a measured distribution of enterprise work.

The first two questions concern capability and assurance. The last two concern deployment and economics. Recurrence is not a logical requirement for a harness to improve an answer; it is what makes the cost of constructing the harness easier to recover.

Trying a problem several times can find an answer missed on the first attempt, but comparing attempts helps only when the system can recognize the better one. A harness can also divide the work, retain intermediate results, retrieve a missing document, or run a precise calculation. These actions change the problem the model must solve. The limit is whether the whole system can perform and check the remaining steps at acceptable quality and cost. Sampling research makes the distinction between generating a correct candidate and selecting it reliably.[10]

Do not assume that these conditions describe most enterprise work, or that a minority of exceptions contains a measured majority of risk. The evidence does not establish those ratios. Measure the addressable share within the actual operation, including the cases that must remain outside automation.

09 / WORKED BUSINESS EXAMPLES

A supplier packet makes the division of labor visible

The objective is a review-ready evidence packet, not an autonomous approval.

Consider a fictional supplier-onboarding process. Its approved procedure requires a supplier profile, a registry record, and a current certificate. Identifiers must agree, registry status must be active, and the supplier must belong to the permitted category. These are illustrative business rules, not a legal standard or a description of a customer deployment.

The controller obtains the three sources and supplies them to a bounded extraction worker. The worker returns specified fields with document and location references. Code checks the output schema, source support, dates, identifiers, and declared category. The controller, not the worker, decides whether the packet is ready for human review.

EXHIBIT 8

Different failures require different responses

Observed condition	Controller response	Result
Required evidence agrees	Apply checks; assemble supported facts.	Ready for human review
Extraction is malformed or unsupported	Return a specific error; allow a bounded retry.	Repair or escalate at the limit
Authentic source records conflict	Preserve the contradiction; do not invent agreement.	Escalate with evidence
Policy is stale or a required source is absent	Stop the normal path; identify the missing prerequisite.	Request correction or human review

Source: author’s reference design. Illustrative outcomes, not live-model performance results.

The distinction between a bad extraction and conflicting evidence is decisive. The first may be repaired by reading again. The second cannot be repaired by asking the model to return an agreeing identifier. A useful harness classifies the failure before deciding whether another model call is appropriate.

The deliverable should present the status, the facts checked, the supporting source locations, unresolved issues, and the policy version. A reviewer should not need to reconstruct the entire execution trace to understand why the case escalated. Traceability supports the work product; it does not substitute for a readable explanation.

A narrow scripted implementation developed for the research dossier exercised clean, repair, and conflict paths. It demonstrates control behavior only. This paper makes no measured claim about the performance of a live model on this workflow. Its purpose is to show exactly what would remain fixed when different runtime models are compared.

Prime42 makes the control system visible

A governed conversation is the interface to an evidence-controlled workflow.

Prime42 is an early-stage watershed intelligence system being built on Lectotype. It connects physical observations, water rights, governing authorities, research, and local operating knowledge around specific places and time periods. Original records remain intact. Normalized values retain their conversion history, and measured, reported, modeled, and contributed evidence remain distinct.

Its Governed Conversational Assistant allows a user to ask a familiar plain-language question, such as how much of a release can be verified downstream and what prevents verification of the remainder. The conversational surface is intentionally ordinary. The intelligence-producing system behind it is not. Before inference, the workflow establishes identity, permissions, geography, time, purpose, and applicable policy. Deterministic tools run the water account and approved rules, and the research layer assembles relevant scientific evidence. Only then does a model receive an admitted evidence package and a bounded task.[17]

EXHIBIT 8B

A governed conversation controls the work before and after inference

Stage	System responsibility	Assistant boundary
Scope	Apply identity, permissions, place, time, purpose, and active policy.	No access outside the admitted scope.
Gather	Retrieve eligible observations, rights, rules, research, and local records.	Cannot prefer a convenient source or relabel its authority.
Calculate	Run approved accounting, conversions, and deterministic rule checks.	Cannot replace a measurement with an estimate or invent a missing value.
Explain	Provide bounded evidence, results, uncertainty, and permitted tools to the model.	May explain, compare, or draft; cannot approve rules or certify delivery.
Verify and record	Check numbers, citations, evidence labels, governing conditions, and preserve the run record.	Release, correct, hold, or escalate according to the acceptance contract.

Source: Lectotype and Prime42 product design materials. Prime42 is under development; this exhibit describes the intended control flow, not independently measured performance.

The assistant can return a supported quantity, a bounded explanation, a scenario, or a refusal. “We cannot verify that yet” is useful when it identifies the missing observation, weak provenance, unresolved balance, conflicting research, or unsatisfied governing condition. A forecast remains a forecast; an estimate does not become a measurement because it appears in a fluent answer.

Post-inference checks reconcile material claims with the admitted package. The run record preserves sources, rules, calculations, assumptions, model and procedure versions, and unresolved issues. This is enterprise-owned intelligence in operational form: the organization retains the evidence, operating procedure, evaluations, and history while the inference provider remains replaceable.

10 / ASSURANCE AND THE WORK PRODUCT

Verify the substance; evaluate the presentation separately

Professional polish and operational correctness are different outcomes, and both matter.

A completed document is not a completed task unless its claims and conclusions meet the acceptance standard. Formatting, fluent explanation, and confident tone can make a work product easier to consume without establishing that it is right. The delivery stage should therefore improve usability without being allowed to conceal unresolved evidence or override a failed check.

The same principle applies inside the harness. A model is not a fully independent sensor of its own errors. Huang and colleagues found that intrinsic self-correction can fail or degrade reasoning without reliable external feedback. Self-Refine found improvements from iterative same-model feedback on other tasks. Together, these results support a task-dependent view of feedback, not a blanket claim that self-critique is either sufficient or useless.[11,12]

EXHIBIT 9

A release scorecard needs three separate views of quality

View	What to measure	What must not stand in for it
Substantive correctness	Supported claims, accurate calculations, completeness, appropriate uncertainty.	Fluency, agreement, or a valid output schema.
Operational reliability	Incorrect accepted outputs, missed exceptions, recovery, outages, and human intervention.	Average score or a falling handoff rate alone.
Professional usability	Clarity, organization, evidence navigation, and remaining reviewer effort.	Aesthetic preference treated as factual correctness.

Source: author’s evaluation framework; feedback evidence [11,12] and professional-deliverable methodology [13].

A deterministic verifier is not automatically independent in the relevant sense. It may encode the same mistaken requirement as the generator. A different model may share the same assumptions. The practical measure is whether the checker rejects the errors that matter, including errors shared by the whole design.

OpenAI's GDPval provides a methodological precedent for blinded expert evaluation of professional work products.[13] A substitution study should separately assess content and natural presentation, using reviewers who do not know the model configuration. "Raters could not tell which model wrote it" is useful only alongside an independent assessment of correctness.

This changes the perception question. The goal is not to imitate the style of a premium model. It is to produce a useful, supported deliverable whose presentation accurately communicates its evidence, uncertainty, and remaining decisions.

11 / THE BUSINESS CASE

Count the cost of the capability, not just the tokens

Compilation moves some expense from each execution into design, evaluation, and maintenance. It does not make that expense disappear.

The relevant financial comparison is total cost per acceptable completion under an explicit latency and risk constraint. A cheap model that requires repeated repairs or extensive review can be expensive at the workflow level. A more costly model can be economical when it avoids orchestration or expert intervention. Both possibilities belong in the business case.

Selling the work raises the verification burden

Sequoia expresses a related commercial shift: "A copilot sells the tool. An autopilot sells the work."[16] Selling a completed outcome transfers more responsibility to the provider. It therefore increases the importance of acceptance contracts, evidence controls, recovery behavior, and reviewable records. A provider that sells the work must be able to show not only that a model produced an answer, but that the system completed the required procedure and handled exceptions correctly.

EXHIBIT 10

Use a lifecycle ledger for both the harness and its comparator

Cost category	Include in the comparison
Build and qualify	Procedure design, human edits, compiler calls, integrations, evaluation cases, and release testing.
Run and recover	Inference, retrieval, tools, retries, infrastructure, stronger-model review, and human escalation.
Maintain and assure	Policy changes, model upgrades, regression tests, incident review, rollback, and remediation.

Source: author’s economic model. No universal savings rate is assumed.

For a fixed planning horizon, a simple first approximation separates fixed costs from variable cost per run. Break-even volume equals the enhanced system’s additional fixed cost divided by its variable saving per run. That expression is useful only when the saving is positive and the costs are reasonably stable over the horizon.

Break-even runs = additional fixed cost ÷ saving per run

The denominator must include review and escalation. A small generator with a flagship reviewer on every case is a hybrid system, not a small-model-only deployment. It may still be the best choice, but the stronger model’s work belongs in the cost and latency record.

Test the business case against lower volume, more exceptions, slower execution, and more frequent policy changes. A procedure that pays back only under perfect reuse is a fragile investment. Track the useful life of each version as carefully as its initial improvement.

The managerial opportunity is a portfolio, not a single standard model. Stable checks can move to code. Repeated interpretation can use bounded model calls. Difficult synthesis can use stronger models. Cases that cannot be adequately checked can remain assisted work. Value comes from assigning each responsibility to the least costly option that meets the requirement.

Economic optimization does not replace mandatory acceptance or risk limits. Some outcomes must be prevented or escalated rather than treated as a tolerable expected-dollar loss.

Workflow redesign creates an economic threshold

The return from AI may not rise smoothly with model quality. The MIT Sloan research shows how modest improvements can leave the existing production design unchanged until a capability threshold makes a longer chain or different job structure viable. At that point, the cost-minimizing organization of work can change discontinuously.[18]

This gives the business case a practical J-curve. Early adoption bears integration, evaluation, and organizational redesign costs before the workflow has removed enough coordination to repay them. A pilot should therefore measure whether it is approaching a reusable operating threshold, not promise that every incremental model improvement will produce an immediate proportional return.

The owned asset is larger than the model

An enterprise can change inference providers without starting over only if the surrounding operating assets remain portable, inspectable, and governed. Those assets turn accumulated experience into an institutional capability rather than provider-specific configuration.

OWNERSHIP CHECK

Five controlled asset classes make intelligence portable

Asset class	Enterprise-controlled artifact	Operational value
Evidence and context	Approved sources, provenance, permissions, and retrieval scope.	Grounding survives a model change.
Procedure	Workflow, tools, deterministic rules, stopping conditions, and escalation paths.	Known work does not have to be rediscovered.
Assurance	Acceptance tests, release thresholds, failure definitions, and incident records.	Quality remains an enterprise decision.
Learning	Run records, exceptions, reviewer corrections, and evaluated trajectories.	Production experience compounds across providers.
Release authority	Approved versions, change owners, rollback points, and review triggers.	Improvement stays separate from uncontrolled production behavior.

Source: author’s synthesis of the model-context-harness ownership frame and the control architecture proposed in this paper.

12 / PROVING THE SUBSTITUTION

Require a fair comparison before scaling

A well-engineered smaller-model system should not be compared only with an intentionally weak larger-model baseline.

A credible test holds the task distribution, acceptance criteria, source access, and permissions constant. It then compares both model classes under both baseline and enhanced harnesses. The baseline should be competent, with a documented tuning budget. The enhanced system changes who controls the work; it should not secretly receive better evidence.

EXHIBIT 11

Four configurations separate model effects from harness effects

	Competent baseline harness	Enhanced cybernetic harness
Smaller model	Establish the starting quality, cost, and error profile.	Measure the added value of explicit control and feedback.
Larger model	Establish the proposed substitution target.	Determine whether the same structure improves the alternative.

Source: author’s comparative evaluation design.

Two further comparisons sharpen the conclusion. Give the smaller baseline additional attempts to determine whether the gain comes merely from more inference. Where feasible, compare a deterministic workflow to establish whether a runtime model is needed at all.

Keep development, configuration selection, and final evaluation separate. The compiler must not see the final test labels. Split realistic cases by entity, template, and time where relevant, rather than scattering near-duplicates across the partitions. Record exact model versions, prompts, procedure versions, tools, retries, latency, cost, and human intervention.

Define an acceptable quality difference in advance. Failing to find a statistically significant difference is not proof of equivalence. Report uncertainty and serious accepted errors, and distinguish justified escalations from missed exceptions. A system that refuses every case can appear safe without being useful; one that accepts everything can appear efficient without being trustworthy.

Finally, separate runtime substitution from compilation provenance. A useful hand-authored harness does not prove that a stronger model made it cheaper to build. A second comparison should record model-assisted design effort, human revisions, and maintenance. That is how the organization learns whether it has built a reusable capability or merely moved expensive work out of view.

An early benchmark can be small and still credible

A young system does not need thousands of production cases to publish useful evidence. It needs a frozen task set, explicit failure definitions, untouched evaluation cases, and complete configuration records.

Prime42 offers a practical starting point because many source records are public and the governed assistant must distinguish supported answers from honest non-answers. The first benchmark should test

the control mechanism rather than claim general intelligence or universal legal and scientific correctness. Domain experts should approve the questions, expected evidence, acceptable uncertainty, and escalation conditions before the final run.

EXHIBIT 12

Seven early-stage benchmarks can test the system without overstating maturity

Benchmark	Initial design	Primary measures
Evidence-bounded answers	About 60 public questions across observations, rights, rules, research, and calculations.	Supported-claim rate; citation accuracy; numeric agreement; evidence-label fidelity.
Exception recognition	About 30 cases with missing, contradictory, stale, or insufficient evidence.	Incorrect acceptance; correct refusal or escalation; stated reason matches the defect.
Permission isolation	About 24 role-and-scope cases mixing public, private, eligible, and ineligible records.	Unauthorized disclosures; eligible-source precision; correct policy selection.
Model-harness matrix	Run the same frozen cases with two model classes under a competent baseline and the enhanced harness.	Acceptable completion; serious accepted errors; latency; tokens; cost; review minutes.
Replay and regression	Re-run about 20 cases after one model, policy, rule, or procedure change.	Reproducibility; intended changes; unintended changes; trace completeness.

Blind work-product review	Domain reviewers score 20–30 paired outputs without knowing the configuration.	Substantive correctness; usefulness; uncertainty communication; remaining reviewer effort.
End-to-end chain economics	About 20 start-to-finish cases spanning several adjacent steps and one defined authority gate.	Acceptable chain completion; human handoffs; intervention minutes; cycle time; serious accepted errors; total cost.

Source: author's proposed early-stage evaluation plan. Sample sizes are practical starting points, not statistical guarantees or completed results.

The first public scorecard should report every denominator, exact model and procedure version, sampling rule, excluded case, retry, escalation, human intervention, and chain boundary. If the task set is too small to support an equivalence claim, report the observed differences and confidence intervals without declaring parity. The useful result is a reproducible mechanism test, a clear error profile, and evidence about how much coordination the governed chain actually removes.

A particularly informative first result would compare direct model answers with governed Prime42 answers on the same public questions. The decisive metrics are not conversational fluency or user preference alone. They are whether each material claim is supported, whether calculations agree with deterministic tools, whether the system refuses unsupported conclusions, and how much review time and total cost are required for an acceptable answer.

13 / THE LEADERSHIP AGENDA

Build a capability portfolio, one qualified workflow at a time

The first objective is not an autonomous enterprise. It is an operating method for deciding where autonomy earns its place.

Begin with a workflow whose owner can describe an acceptable result and identify the evidence behind it. Assign three responsibilities explicitly: a business owner for the acceptance standard, an engineering owner for execution and recovery, and an assurance owner for independent evaluation and release. One person may fill more than one role; the decisions should remain distinguishable.

Redesign jobs as well as workflows

A production chain changes work allocation, not merely software behavior. As the system takes responsibility for adjacent, checkable steps, people may spend less time transferring context and more time defining acceptance standards, resolving exceptions, improving the procedure, and exercising authority at consequential boundaries.

Treat that redesign as part of the implementation. Name the work that disappears, the work that moves, the judgment that remains human, and the new responsibility for evaluation and release. Otherwise the organization can build an efficient technical chain while leaving the old coordination structure intact around it.

EXHIBIT 13

A 90-day starting sequence turns the thesis into a decision

Phase	Work to complete	Decision gate
Days 1–30Choose and specify	Bound one workflow; document evidence, exceptions, baseline costs, and acceptance rules.	Is the task checkable, and is enough volume plausibly reusable?
Days 31–60Build and challenge	Implement a bounded loop; compare configurations; test contradictory, unfamiliar, and stale inputs.	Does the system meet quality and recovery requirements on untouched cases?
Days 61–90Pilot and decide	Operate within a limited scope; measure review effort, accepted errors, cost, and change burden.	Scale, redesign, retain as assisted work, or stop.

Source: author’s proposed adoption sequence. Timing is a planning framework, not a deployment or return guarantee.

Buy generic execution infrastructure where it satisfies requirements; retain ownership of business policies, evidence authority, acceptance tests, and release decisions. The reusable value lies in that operating knowledge, not merely in the number of tools connected to a model.

Revisit the portfolio when models, prices, policies, or workloads change. Remove compensatory structure that no longer earns its cost. Preserve controls whose purpose is accountability. Treat each successful workflow as a versioned operating asset, not as proof that every neighboring task is now safe to automate.

Use powerful models where the problem is new. Preserve what has been learned where the work repeats. Own the system that selects, constrains, checks, and improves the intelligence.

That is the business case for compiling capability. The published evidence does not establish universal model-class equivalence, but it does establish mechanisms worth engineering and testing. The next advantage need not come only from buying a more capable model. It can come from owning the intelligence-producing system around the model: a system that asks the model to do less of the wrong work, preserves what the organization has learned, and makes the remaining work reliable.

NOTES AND SOURCES

Evidence behind the argument

Publications support the specific mechanisms and findings cited in the text. Conceptual exhibits, the reference architecture, and the adoption framework are the author’s synthesis. Company-reported outcomes have not been independently reproduced.

1. Khattab, O., et al. DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines. ICLR 2024; arXiv:2310.03714.

Program and prompt optimization are distinct from workflow-topology discovery and optional weight optimization.

2. Singhvi, A., et al. DSPy Assertions: Computational Constraints for Self-Refining Language Model Pipelines. arXiv:2312.13382, 2023; revised 2024.

Cited for the research mechanism, not as a claim about an unchanged present-day library API.

3. Stripe. Minions: Stripe’s one-shot, end-to-end coding agents; Part 2. February 9 and 19, 2026.

Mechanism and usage statements are restricted to verified official passages; full article-body retrieval was partial. Weekly volume is company-reported and includes human review.

4. Winkelmeyer, R. / Salesforce. Introducing Hybrid Reasoning with Agent Script. October 2025.

Documents compilation to an Agent Graph and execution through Atlas. No model-tier substitution result is reported.

5. Dalley, C. / Intercom. Announcing major updates to Procedures and Simulations: Enabling Fin to handle complex queries. February 25, 2026.

Product and customer examples establish procedure, control, and revalidation mechanisms; not a controlled model comparison.

6. Google DeepMind, AlphaEvolve team. AlphaEvolve: A Gemini-powered coding agent for designing advanced algorithms. May 14, 2025.

The 0.7% compute-recovery figure is Google’s reported production average, not an independently reproduced result in this paper.

7. OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition. 2026, “Availability and pricing.”

Used only for Luna-Terra-Sol product-role terminology. Functional classes do not imply cross-vendor benchmark equivalence.

NOTES AND SOURCES

Sources and publication scope

8. Rajasekaran, P. / Anthropic. Harness design for long-running application development. March 24, 2026.

Engineering observations across model generations. They support testing model–harness interactions, not a universal decrease in harness value.

9. OpenAI; attributed SafetyKit and Hypha statements. Introducing the Agents API. September 10, 2026, customer statements.

The cost and failure reductions are customer reports hosted by the vendor. Underlying logs, sample sizes, and controlled counterfactuals are not disclosed.

10. Brown, B., et al. Large Language Monkeys: Scaling Inference Compute with Repeated Sampling. arXiv:2407.21787, 2024.

Candidate coverage and reliable selection are different outcomes. Sampling results do not bound every interactive architecture.

11. Huang, J., et al. Large Language Models Cannot Self-Correct Reasoning Yet. arXiv:2310.01798, 2023.

Negative results concern intrinsic self-correction in the tested settings, not all possible feedback systems.

12. Madaan, A., et al. Self-Refine: Iterative Refinement with Self-Feedback. NeurIPS 2023; arXiv:2303.17651.

Positive task-specific results do not establish independence between generator and checker errors.

13. OpenAI. Measuring the performance of our models on real-world tasks. GDPval, September 25, 2025.

Cited as a precedent for blinded expert assessment of professional deliverables, not as evidence of this paper's substitution hypothesis.

1. Huang, S. / Sequoia Capital. Own Your Intelligence: A How-To Guide. August 19, 2026.

Ownership framing; not evidence of Lectotype performance.

1. Chase, H. / Sequoia Capital. Owning Your Intelligence Starts With the Harness. 2026.

Model-context-harness framing, portability, observability, and production learning.

1. Bek, J. / Sequoia Capital. Services: The New Software. March 5, 2026.

Commercial tool/work distinction; verification implications are the author's interpretation.

1. Lectotype / Prime42. Prime42 revised narration, technical deep-dive narration, and Great Salt Lake live demonstration script. September–October 2026.

Product-design materials describing intended control flow; not independent validation or completed benchmark results.

1. Demirer, M., Horton, J., Immorlica, N., Lucier, B., and Shahidi, P. Chaining Tasks, Redefining Work: A Theory of AI Automation. NBER Working Paper 34859, 2026; summarized by MIT Sloan on April 22, 2026.

Cited for the economics of sequential work, contiguous AI chains, coordination costs, job redesign, and capability thresholds. It is not evidence of Lectotype performance, a governance result, or a smaller-model substitution result.