

Agent Chronoception: Temporal Validity and Useful Persistence

Aojie Yuan*
University of Southern California
aojieyua@usc.edu

Zijian Su
University of Michigan
simoon@umich.edu

Abstract

Personal assistants, coding agents, and collaborating agents must decide not only how to solve a task, but whether further work is worth the user’s wait. Early stopping, prolonged activity, and accurate duration reports do not by themselves reveal that competence. We argue that agent chronoception should be evaluated through useful persistence: continuing, revising, waiting, or delivering according to the value of feasible work under an explicit user contract. We propose three complementary requirements: stable choices under decision-equivalent temporal changes; selective adaptation when timing changes alter plan value; and preservation of temporal validity through deliberation, execution, and handoff. An exact counterexample shows why punctuality and invariance alone are insufficient. Controlled coding and browser screens examine budget aging and the cost of replanning. In valid matched single-decision slowdown/recovery comparisons, model controllers deliver the same quality as an equally informed scheduler while taking longer; an offline sensitivity analysis makes the dependence on waiting cost explicit. All 36 controller conditions were attempted: 34 yielded valid timed outcomes and two failed the plan schema. A separate 32-trajectory multistage study compares immediate and delayed scheduling with model replanning and conditional runtime control, charging all within-trajectory time. Constructed tests clarify diagnostic distinctions rather than model capability. We propose a joint evaluation standard, a portable temporal record, and an attribution protocol for testing when persistence serves the user. Long-horizon transfer, personalized waiting preferences, and benefits from reinforcement learning remain open empirical questions.

Keywords: agent chronoception; useful persistence; personal agents; coding agents; long-horizon evaluation; metareasoning.

Running title: Agent Chronoception and Useful Persistence

Position. Evidence for agent chronoception should include both stability under decision-equivalent temporal changes and useful adaptation when temporal changes alter the decision problem. Duration forecasts should be evaluated for calibration and decision value.

1 More time creates a decision problem

A request to “keep improving this until my meeting” delegates a sequence of choices: what to investigate, when to verify, when to deliver a usable checkpoint, and whether further work is worth the wait. A larger budget expands what may be attempted; it does not tell the agent which attempts are valuable. A coding agent may already have a working patch, unresolved tests, and time remaining before delivery. Expressing that time in milliseconds rather than seconds should not change which test is worth running. A sudden increase in the duration of the full test suite may change that choice. If the service later recovers, the earlier conservative plan may no longer be appropriate. These cases pose a more discriminating question than whether the agent mentions time: does its behavior track the temporal relationships that matter to the task?

We use *agent chronoception* operationally, for estimating durations and using temporal evidence to regulate action. This definition makes no claim about phenomenal experience, a biological clock, or an inaccessible internal mechanism. A system may implement the capability through an external predictor and scheduler. If so, the result belongs to the system; it should not be attributed to the language model alone.

*Corresponding author: aojieyua@usc.edu. ORCID: 0009-0007-4643-230X.

Useful persistence, not persistence for its own sake. We use *useful persistence* as an organizing term for retaining a delegation only while its permitted continuation, monitoring, or waiting serves the user’s contract. It includes timely delivery and justified termination. Apparent “agent laziness” is therefore a behavioral symptom to diagnose, not a motive or a single mechanism: the task may be saturated, the interface may have ended a turn, information may be missing, or a valuable continuation may have been abandoned. An early-stop failure requires evidence of an authorized, feasible continuation with positive expected value at the decision point, including its decision and verification costs. A successful continuation discovered only in hindsight is insufficient.

Our disagreement with outcome-only evaluation. Deadline adherence and task quality remain essential. We argue that they are insufficient to establish *transferable* chronoception, because a fixed schedule can succeed on a familiar timing regime. Conversely, invariance alone is insufficient: an agent that ignores every clock is invariant. The two requirements must be evaluated together, on tasks where temporal changes sometimes preserve and sometimes reverse the value ordering of actions.

Contributions: a standard, a record, and a diagnosis. We advance three reusable components for studying agent chronoception.

1. **A joint evaluation standard.** Require useful stability under decision-equivalent temporal transformations and selective adaptation under consequential changes. An exact constructed example separates this requirement from punctuality and invariance alone.
2. **A record of temporal validity across boundaries.** Specify what must be retained from observation through deliberation, dispatch, and handoff: the root delivery obligation, the age and uncertainty of timing evidence, integration costs, and the artifact actually delivered. This is an evaluation interface, not a claim to invent deadline propagation.
3. **An attribution protocol.** Compare refreshed timing evidence, post-decision admission checks, and equally informed schedulers with model replanning. Distinguish missing evidence, expired decisions, mistaken forecasts, and costly deliberation before prescribing learning.

The first component is supported by a constructive argument; the latter two organize a testable methodology. Short coding and browser screens exercise parts of that methodology and expose its limitations. Their results do not validate autonomous long-horizon transfer, a new training algorithm, or a model-internal clock.

Why this matters for delegated work. The relevant unit of success is a useful handoff. A coding agent can spend its entire allowance testing an obsolete patch; another can stop early because the requested change is already verified. Their elapsed runtimes alone rank neither behavior. The same distinction applies when an agent waits for a remote job: another message or another tool call may add no value, while preserving the job identifier and returning when evidence becomes available may be essential. Chronoception should help discriminate between these cases, not make every interaction longer.

The distinction also matters to users whose requirements change. A researcher may value a quick, usable result before a meeting and a more thorough revision afterward. A policy that always spends the maximum budget disregards the first preference; a policy that always returns the first plausible result disregards the second. We therefore tie temporal behavior to a declared contract. A deadline is a constraint, a duration forecast is evidence, and a preference over waiting is part of the objective. Treating them as interchangeable clock tokens obscures what an agent should learn.

2 What counts as useful additional time?

A quality-versus-budget curve is informative only when a task offers meaningful opportunities for improvement. A correct solution to a simple task may already be available at the smallest budget. A flat curve then indicates saturation rather than failed chronoception. The development screens in Appendix E illustrate this design problem: their elementary repairs and local form updates do not require a richer strategy when more time is available. Repeating those tasks at larger scale would not establish the proposed capability.

Distinguish opportunity from obligation. A longer allowance permits additional work; it need not require waiting until the deadline. A request for continuous monitoring is different because observation coverage is itself part of the task. Evaluations should declare whether the contract requires an artifact, a period of coverage, or

both. Otherwise, a prompt to “use the full time” can penalize appropriate early delivery and reward unnecessary activity. The same problem arises when the only reward for a long-horizon agent is survival or action count.

Evaluate what reaches the user. More computation can improve a private draft while delaying the last usable artifact. Conversely, an early checkpoint can provide value while a longer verification process continues. Measure the artifact actually delivered at prespecified inspection times and report regressions as well as gains. Do not construct an optimistic curve by selecting the best historical checkpoint with a private evaluator. Such a curve describes an unavailable selection oracle rather than the deployed agent.

Separate extra time from a different decision problem. Increasing a deadline, slowing a tool, and changing a user’s cost of waiting are distinct interventions. A longer deadline may admit a full test suite; a slower suite may make a targeted check preferable; a user who needs an immediate draft may favor delivery before either check. A useful temporal evaluation makes these distinctions observable while keeping other task evidence matched. Unit conversion is a negative control within that design, not its primary practical motivation.

Use disagreements to locate the bottleneck. Two systems can finish equally often and obtain similar final scores while making different decisions after a delay change. This is a hypothesis for the proposed study, not a result established by our pilot. If an explicit scheduler corrects the difference, the remedy may be architectural. If accurate forecasts are available but ignored, the issue is their use in planning. If neither system has a useful continuation, further training on persistence is misplaced. The purpose of chronoception evaluation is to make these explanations distinguishable.

3 Three times, one testable relationship

Retain three distinct measurements. τ_{wall} is monotonic elapsed time measured outside the policy. τ_{step} indexes actual policy decisions and executed actions, not serialized messages or tokens. τ_{self} denotes explicit retrospective or prospective duration estimates. A reported estimate is an observable output, not direct access to an internal belief.

A serial execution ledger separates inference, tool execution, and other waiting intervals. These intervals must account for elapsed time without overlap. Concurrent jobs instead require an event graph: adding two overlapping runtimes overstates the user’s wait. Counting actions is also insufficient, because an identical sequence can take very different amounts of time. Appendix A gives the accounting identity.

Before action a_k , a temporal predictor supplies a conditional duration distribution $p_\phi(\Delta \mid h_k, a_k)$. After execution, the measured duration updates the forecast for later actions. Remaining-plan time also depends on future choices, retries, and parallelism; it cannot generally be obtained by summing independent marginal forecasts. We record the declared plan and forecast horizon with every prediction.

4 Relationship to temporal control and agent learning

Timely Machine already couples wall-clock budgets and tool latency with temporal planning and RL [5]. On the Clock studies harness feedback, enforcement, duration estimation, and budget-aware RL, separating deadline adherence from productive time use [6]. Thus, duration prediction, clock access, and learning a budget-conditioned strategy cannot be claimed as our inventions.

TicToc examines time-sensitive tool choices [7]; interactive planning incorporates user interaction [8]; budget control and metareasoning value additional computation [9, 11, 12]. Real-time RL studies learned planning budgets [10]. Semi-Markov options model variable-duration actions [15], and scale-invariant future-time representations have a substantial prior history [17]. Long-horizon trajectory selection and memory training address further causes of unreliable persistence [13, 14].

Chronocept studies how the validity of facts and passages evolves over time [3]. Our use of temporal validity concerns a decision: whether the evidence and remaining budget that justified a continuation still support it at dispatch or handoff. We do not introduce chronoception as a general AI concept or replace factual-validity evaluation.

Adjacent evaluation research also limits the novelty claim. Gaia2 evaluates agents in asynchronous environments with temporal constraints and changing events [18]. Tempora studies time-contingent utility in online

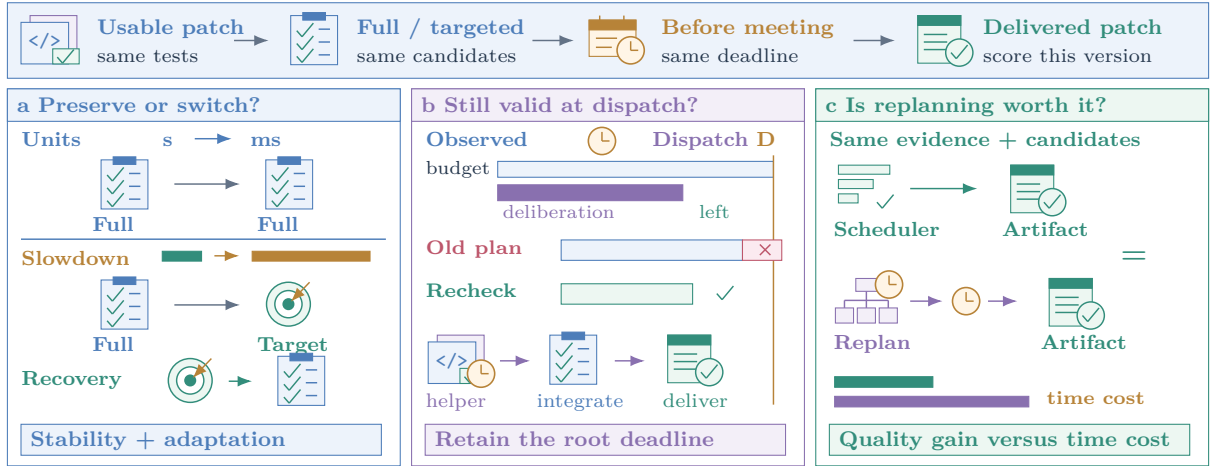


Figure 1: Three complementary contributions shown through schematic rollouts. (a) Joint tests require stability under unit conversion and selective adaptation under slowdown and recovery. (b) A choice can expire during deliberation; handoffs must reserve integration time within the root deadline. (c) Equally informed controllers must be compared on delivered quality and total time cost. Icons depict artifacts, tests, dispatch, and delivery; bars in this figure are qualitative, not measurements.

test-time adaptation [19]. Thus, dynamic tasks and latency-sensitive utility are established directions. Our proposed contribution is a controlled diagnostic comparison within an agent task: identify when timing should leave a useful choice unchanged and when it should alter that choice. We do not infer that these earlier frameworks cannot support such a comparison.

AgentTime evaluates duration-following, runtime forecasts, and retrospective estimates in native agent harnesses, and explicitly distinguishes matching a requested runtime from continued useful work [4]. Thus, early stopping, timing reports, and the inadequacy of runtime matching are not sufficient novelty claims here. Our proposed unit of comparison is a matched decision and its downstream delivery: vary whether temporal evidence remains valid and whether the preferred continuation should change, while accounting for the cost of making that decision.

Recent meta-reasoning work already allocates investigations across long research trajectories [1], and adaptive reasoning methods learn when prior reasoning remains sufficient [2]. Useful persistence does not claim those control problems as new. Our proposed contribution is their temporal audit: whether the evidence supporting continuation remains valid by dispatch and across delegation, and whether the complete decision process improves delivery over equally informed simpler control.

The question left by recent timing work. Recent budget-aware systems already make punctuality and productive time use explicit [5, 6]. Repeating that distinction would not establish a new contribution. Our narrower question is whether a temporal strategy survives interventions that separate irrelevant numerical changes from consequential changes in execution. In particular, a system that succeeds under one budget or familiar latency profile may still react to a unit conversion, fail to update after a slowdown, or retain a conservative strategy after recovery. These are hypotheses to test, not deficiencies established for every existing system.

The proposed standard is a *joint experimental requirement*: test equivalent transformations, consequential changes, and interventions on explicit forecasts within the same controlled agent interface. Its purpose is to distinguish sensitivity to numeric cues from adaptation to execution dynamics, and accurate predictions from predictions that improve decisions. Existing systems may already meet this standard; applying the tests would establish whether they do. The individual ingredients have precedents, while the empirical value of combining them remains a question for the proposed study.

5 Scope: chronoception across delegated work

Our position concerns *decision-relevant temporal competence under delegation*: using observed durations and uncertain forecasts to preserve a user’s delivery requirements across continuation, interruption, and handoff.

Established ingredient	Already supported by prior work	Proposed discriminating test
Clock and budget-aware policy	Time feedback, deadline control, temporal RL	Same task under equivalent versus consequential timing interventions
Duration prediction	Forecasting operation and completion time	Replace or freeze forecasts while holding controller and information access fixed
Temporal scale representations	Scale-invariant time and value models	Explicit equivalence conditions, followed by held-out LLM-agent tests
Metareasoning and stopping	Value of computation and variable-duration options	Correct adaptation without increased abandonment or useless persistence

Table 1: A novelty boundary, not a claim of missing features in every related system. Our proposed tests and training objective remain prospective.

Personal assistants, coding agents, and collaborating agents are deployment settings for this question, not separate capabilities that this paper has validated. Long horizon denotes repeated opportunities to revise a plan as information and dependencies change; an artificially extended runtime alone does not establish it.

From a local clock to a shared obligation. A personal assistant preparing a meeting brief must distinguish an available draft from a finished background investigation. A coding agent must distinguish a patch from the verified version that a reviewer can use. A coordinating agent must leave time to integrate a helper’s result. These cases share an obligation whose satisfaction depends on what another participant can actually use and when. A progress message, a locally completed subtask, and a usable handoff are different events.

Temporal assistance and coordination have substantial precedents. PTIME addressed personalized time management and deployment within a personal assistant [20]. A recent engineering example propagates a shared elapsed clock and budget to a lead agent and its helpers [21]. Our proposed contribution is not clock propagation or the invention of scheduling. It is to test whether selective temporal competence survives composition: do locally reasonable choices still support the original delivery obligation when tasks are delegated, interrupted, or recombined?

A compositional failure to test. Suppose a helper is asked to return evidence before a deadline, but the parent still needs to verify and integrate it. Giving the helper the entire remaining budget can make both local compliance and global failure possible. With current time t , user deadline D , and reserved integration duration r , a proposed handoff deadline d_h must leave $d_h + r \leq D$. If a forecast distribution for helper completion is available, a further feasibility check is $\Pr(T_h \leq d_h - t) \geq 1 - \epsilon$. These are elementary scheduling constraints, not a new theorem. Their assumptions must be exposed: an estimated r is not a guarantee, and multiple uncertain dependencies cannot be reduced to independent means without justification. The empirical question is whether an agent retains and revises such obligations when the plan changes.

Three falsifiable extensions. First, under a *user-contract intervention*, keep the task and tools fixed but vary when a usable checkpoint is valuable. Measure delivery quality at the requested time, unnecessary interruptions, and work after cancellation. Second, under a *handoff intervention*, preserve the end-to-end deadline while varying a helper’s delay and the parent’s integration reserve. Compare global delivery with local on-time rates; report critical-path delay and redundant work. Third, under a *recovery intervention*, restore a delayed service after observable evidence of the slowdown. Test whether a conservative plan is reconsidered when it no longer maximizes useful work. Correct behavior need not always involve switching, delegation, or using the entire budget.

A shared experimental object. Each extension should record the original contract, permitted observations, dependency state, pre-action duration forecasts, submitted artifact version, and actual handoff time. Equivalent clock representations must preserve all affected fields. Consequential changes should sometimes preserve and sometimes reverse the value ordering of feasible choices. Freeze the worker and give a rule-based scheduler the same observations before attributing value to a learned controller. Hold out dependency structures and task families, not merely prompts or random seeds. For coding and browser tasks, existing executable checks provide an initial outcome measure; personal-agent preferences additionally require user-grounded validation.

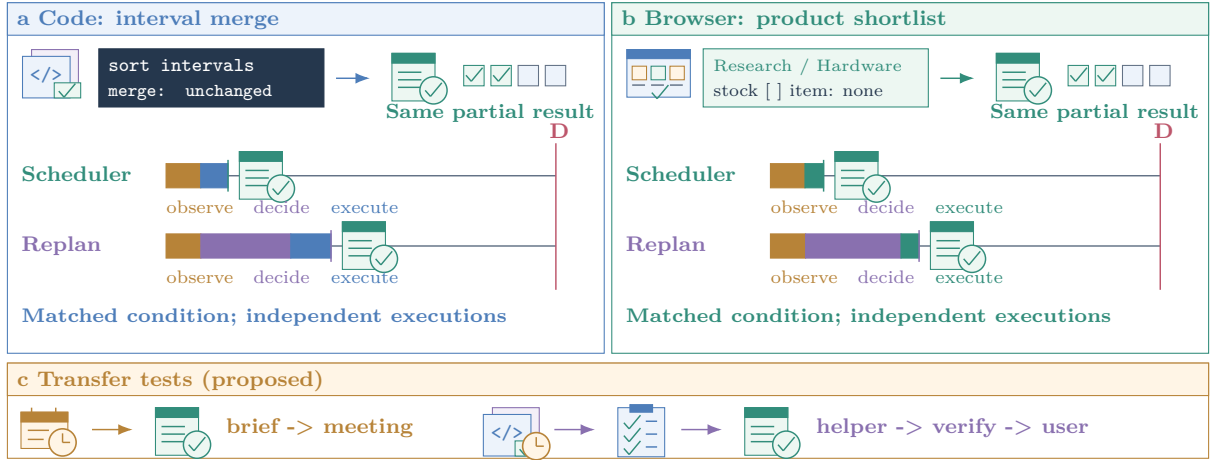


Figure 2: Recorded slowdown examples and prospective transfer. Panels (a,b) show matched repetition-zero scheduler and replanning runs on interval merging and catalog filtering. Both controllers selected PARTIAL and passed the same two checks; unmarked boxes are unmet enhancements. Bars use the same elapsed-time scale and actual observation, dispatch, and finish timestamps, with deadline D at the common right edge. Execution includes verification and cleanup; setup is excluded. These selected independent executions illustrate cost accounting, not a causal estimate or general superiority claim. Panel (c) is a proposed extension, not an executed rollout.

A portable temporal record. For each decision, retain a record

$$\mathcal{R}_k = (\mathcal{C}, t_{\text{obs}}, t_{\text{dispatch}}, \hat{P}_k, G_k, v_k),$$

where $\mathcal{C}$ records the root deadline, mandatory requirements, and waiting preferences; $\hat{P}_k$ is the forecast with its conditioning evidence; G_k records pending dependencies and integration work; and v_k identifies the last delivered artifact. This record joins familiar scheduling and provenance fields for an auditable comparison, rather than defining a new scheduling algorithm. Cross-host timestamps require an explicit mapping to the same reference clock, with synchronization uncertainty recorded; unrelated monotonic clocks cannot be subtracted. Missing fields are observable interface limitations, not evidence of a model’s inability to use information it never received.

What would generalization mean? The transferable object is the relationship between a temporal intervention, a warranted decision, and a usable delivery, rather than a particular tool name. Coding, browsing, personal assistance, and delegation instantiate different actions and utility functions (Figure 2). Applying one record format across them makes comparisons possible; it does not make their scores interchangeable. A claim of transfer requires held-out mechanics and dependency structures with independently validated outcome measures. We propose this common test interface so that future methods can be compared without requiring a particular agent architecture or training recipe.

Boundaries that protect the claim. Time pressure cannot authorize omitting mandatory checks, changing permissions, or silently substituting an incomplete result. Hard requirements define feasible plans before waiting cost ranks them. Our exploratory graded-task diagnostic deliberately permits partial completion and therefore cannot validate this stronger constraint. Nor can single-decision selections establish durable memory, complex planning, or agent-to-agent coordination. Those are prospective transfer tests. A useful position paper contributes the operational distinction, a counterexample to inadequate evaluation, and a reproducible route to falsification; it need not claim a universal new architecture or assume that RL is the best solution.

6 Separate equivalence from temporal change

A temporal intervention must specify what changes and what remains fixed. The user contract includes the deadline, resource cap, delivery rules, stopping conditions, and preferences over quality and waiting. The agent receives permitted observations; it does not receive the experimenter’s hidden duration law.

Representation equivalence. Expressing the same deadline in seconds or milliseconds should preserve the decision problem. Convert every affected timestamp and tool argument consistently, then compare decisions in common units. The goal is agreement about which useful work to do, not identical wording or an identical patch. Different near-optimal actions can be equally defensible.

Conditional physical equivalence. Slowing an entire simulated world can also preserve a decision problem, but only if deadlines, waiting costs, discounting, and other time-dependent quantities transform consistently. Slowing tools alone generally does not. We treat full rescaling as a simulator control with auditable assumptions, not as a claim about ordinary deployment. Formal conditions appear in Appendix A.

Consequential latency interventions. Change one tool’s delay distribution while holding the deadline, task semantics, and user preferences fixed. Longer delay need not always imply a different action: the same test may remain indispensable. Include both preference-reversing and preference-preserving pairs. Establish the expected ranking of a disclosed feasible plan set with independent branch replications; label ambiguous pairs as uncertain rather than forcing a switch label.

Adaptation and recovery. Introduce an unannounced latency change only after a shared checkpoint. At that instant, policies with identical observations should not be expected to know the hidden change. Measure adaptation after observable evidence arrives, and include recovery to the original regime. Track the number of observations and the utility loss during updating. Do not reward clairvoyance or infer adaptation from a changed prompt alone.

The user remains part of time. “Deliver before my meeting” and “improve in the background” are distinct contracts even at the same deadline. A contract intervention changes utility, so it is not an invariance test. Delivering a usable checkpoint and ending the delegation remain separate actions. We retain user waiting costs as a downstream test of chronoception, rather than substituting persistence itself for the central research question.

7 Why single temporal scores are insufficient

Consider a constructed one-decision environment with two verification-and-delivery plans. The full plan produces quality 0.90; the targeted plan produces quality 0.80. Their stipulated times and outcomes are known to an informed selector. Utility is $U = q - 0.02\Delta$, with deadline ten time units. The full plan takes four units in the fast regime and nine in the slow regime; the targeted plan always takes two. All plans meet the deadline. These are authored assumptions, not measured software quality or LLM results.

Plan	Fast regime		Slow regime	
	Duration	Utility	Duration	Utility
Full verification and delivery	4	0.82	9	0.72
Targeted verification and delivery	2	0.76	2	0.76

Table 2: Exact constructed witness. The preferred plan reverses solely because the full plan becomes slower. Every plan is punctual, so deadline success alone does not distinguish the strategies.

What the example establishes. A policy that always runs full verification meets every deadline and can ignore irrelevant changes in time units. Yet it loses utility in the slow regime. Always choosing the targeted plan loses utility in the fast regime. Randomizing between the two without using the regime cannot eliminate this loss: its average regret is at least 0.02 under equally weighted regimes. An informed selector uses the full plan when fast and the targeted plan when slow, avoiding that loss. Appendix A gives the exact calculation. Punctuality and invariance therefore cannot, even jointly, distinguish a timing-insensitive policy from useful adaptation.

The claim is deliberately limited: it separates evaluation criteria rather than establishing a difficult new control theorem. Before an unannounced regime becomes observable, the informed selector is not an attainable baseline. For adaptation experiments, evaluate relative to information available after a common diagnostic observation or report the value of privileged information separately. Merely switching whenever any number changes also fails: a seconds-to-milliseconds conversion leaves the fast-regime preference unchanged.

Report sensitivity and decision loss together. Compare how often an agent changes its canonical plan on equivalent pairs, alongside the utility lost relative to the disclosed feasible plans in each state. Report preference-reversing, preference-preserving, and ambiguous cases separately. A stable but consistently poor policy should not receive credit for adaptation. Nor should a policy be penalized merely for choosing a different equally useful plan. The formal diagnostics in Appendix A are complementary measurements, not ingredients for an arbitrary single “chronoception score.”

8 Test whether forecasts affect useful action

A calibrated prediction can be ignored, and a useful scheduler can succeed without an explicit learned predictor. We therefore propose forecast interventions alongside behavioral tests. Freeze a worker and controller interface, then compare an online learned forecast, its frozen pre-shift version, a moving-average estimate, and no forecast. Keep clocks, observed duration histories, task evidence, action limits, and prompt scaffolding matched. Include a matched-size shuffled-forecast negative control with its potential distribution mismatch disclosed.

A distribution-informed forecast is an upper-reference condition: it may know the generating delay law but must not reveal the realized future random draw. Compare delivered quality, deadline risk, and decision utility, not just error in a narration field. Randomized forecast substitution estimates the effect of that *exposed forecast channel in this controller*. It does not establish that a natural-language self-report mediates an opaque model’s internal reasoning. If the policy recomputes durations from history, a weak substitution effect may reflect redundancy rather than an absence of temporal competence.

Evaluate what the user actually receives by a prespecified deadline or inspection time. Include delivered quality, waiting costs, charged computation, and disruptive requests for user input. A private evaluator must not select the best intermediate artifact retrospectively and credit it as though the agent had delivered it. Synthetic preferences make a controlled comparison possible; they are not measurements of human satisfaction.

The practical continuation question is whether a feasible sequence of further work is worth more than ending now. That sequence may include testing, waiting for a pending job, or delivering a checkpoint before continuing. One unpromising next action does not prove that every continuation is useless. Appendix A records the corresponding utility and continuation definitions.

9 A prospective learning program

We propose *paired temporal training*, beginning with a frozen worker and a small controller. Collect real executions with pre-action duration forecasts, measured disjoint timing intervals, and independently scored delivered artifacts. Fork resettable states under equivalent representations and under controlled latency changes. Branch data must include unsuccessful and truncated trajectories; timeout observations are right-censored durations, not completed calls with a short runtime.

A candidate learning objective has three components: reward useful delivery, score duration forecasts against observed outcomes, and encourage consistent decisions on validated equivalent pairs. The consistency term must never force agreement when a consequential change makes another plan better. A timeout is a censored observation, not a completed operation with a short runtime. Appendix A specifies the candidate objective. For closed API models, sampled decisions can be evaluated, but inaccessible policy probabilities cannot be treated as an available training interface.

For changed dynamics, collect repeated candidate-plan outcomes and train value estimates or policy preferences on the observed utility differences. Do not reward change for its own sake, duration consumption, or agreement with a self-generated forecast. RL is a candidate optimizer, not a contribution by name. Compare ordinary utility-only RL, identical-data supervised training, forecast-only training, consistency-only training, and the combined objective under matched collection and optimization budgets.

A reason to test transfer. Paired training might reduce reliance on familiar numeric ranges and separate duration estimation from task-specific action choice. It might also over-regularize useful decisions, teach only arithmetic conversion, or lose to a simple estimator. These are competing hypotheses. Hold out repository families, tool identities where feasible, latency distributions, and change-point schedules; fresh prompts from the same task do not demonstrate transfer.

10 Decisions must remain valid through deliberation

We focus on whether a choice remains useful between observing temporal evidence and executing the resulting plan. The same accounting applies to controller self-revision, but recursive improvement is a downstream research direction rather than a claim of this paper (Appendix K).

Validity must survive deliberation and delegation. A budget observation is valid at a particular instant, not for the entire ensuing decision. If the remaining budget at observation is b_t and deliberation takes c , the executable budget is $b_t - c$. A candidate with predicted work duration $\hat{d}$ and integration reserve r must be assessed at dispatch against

$$\hat{d} + r \leq D_{\text{root}} - t_{\text{dispatch}},$$

where the right-hand side uses the original delivery deadline. This is a feasibility check under a forecast, not a guarantee about stochastic execution. A helper’s local clock must not silently restart that deadline. Deadline propagation is established systems practice [26]; inherited stale constraints are also an existing agent-research problem [27]. Our proposed contribution is therefore a discriminating evaluation: separate failure to update available temporal evidence, loss of evidence during handoff, and unavoidable uncertainty in execution duration. An extra clock field or an ordinary timeout wrapper alone would not establish novelty.

Decision validity beyond expiration. A decision can remain feasible yet cease to be useful when a previously slow service recovers or a better continuation becomes available. A prospective paired test should therefore distinguish expiration from a reversal in the ranking of feasible plans. Give each controller the same observed slowdown or recovery, charge for reconsideration, and compare retaining the plan, a cheap current-state scheduler, model replanning, and a restricted conditional plan evaluated at dispatch. Include unchanged-state and equivalent-unit controls, and score useful improvements forgone alongside late starts. This extends our diagnostic beyond the single-candidate screens. Appendix L exercises a narrow disclosed-delay, prepared-plan version; learning from uncertain observations and long-horizon transfer remain open. Concurrent planning and execution, and allocation of planning effort under deadlines, already have formal metareasoning treatments [28, 29]. The research question is whether the proposed comparisons distinguish an agent’s selective adaptation from the effect of its scheduler, not whether planning consumes time.

What the completed screen supports. The 128-condition screen in Appendix J does not establish a timestamp-prompt advantage: all 64 tight-budget decisions retain the existing checkpoint, and 59 of 64 wider-budget decisions deliver an upgrade on time. One request expires during inference before a late revision; no chosen revision is infeasible at request under the supplied forecasts. Moreover, 37 checkpoint-retention responses arrive after the deadline even though the checkpoint was already delivered. These observations motivate separate reporting of artifact availability, control-response latency, and dispatch feasibility. They do not identify a new mechanism or establish stable intervention effects.

Do not conflate a bad decision with an expired decision. Record feasibility at two instants: when a decision request observes the clock, and when its chosen action is ready to dispatch. A request can be feasible under its supplied forecasts yet become infeasible while the model computes. That event tests the composition of inference and execution, rather than establishing that the model ignored information available in its prompt. Conversely, admitting a plan already infeasible under the exposed estimates is a distinct forecast-use error. Forecast error can affect either category, so retain actual durations alongside both classifications. A runtime admission check can prevent some wasted actions, but cannot recover time already spent deciding. Its benefit should therefore be evaluated together with missed useful continuations, rather than with deadline violations alone.

A useful falsification compares an old snapshot plus its explicit age with the equivalent refreshed remainder, then compares the same model-generated handoff with and without an independently preserved original deadline. Include a post-decision admission check as a cheap systems baseline. If all information-equivalent conditions yield appropriate decisions and handoffs retain the deadline, the hypothesized failure is absent on those tasks. If a deliberately stripped handoff fails, attribute that result to missing information rather than inability to use it. Timed traces must include the cost of producing and reading the handoff, together with any injected waiting, and score only artifacts actually delivered before the root deadline.

Test	Intervention and control	Report separately
Equivalent units	Seconds / milliseconds; canonicalized tools	Invalid units; action-distribution divergence; utility
Equivalent scale	Simulator-only joint scaling with proven return preservation	Policy disagreement; audit of all scaled quantities
Selective slowdown	One tool slows; task facts fixed	Candidate regret on ranking-reversing and preserving pairs
Shift and recovery	Unannounced change, then recovery	Post-observation forecast calibration; adaptation loss
Forecast substitution	Learned / frozen / simple / absent forecasts	Forecast-channel effect on quality, delay, and violations
Long-horizon transfer	New dependency chains and context resets	Retained timing state; completed stages; useful delivery

Table 3: Prospective tests. These are confirmatory study designs. The later selection diagnostic exercises a narrow disclosed-estimate version of unit and latency interventions; it does not complete these tests. Existing microtasks in Appendix C test narrower interface properties.

11 Experiments that can reject the position

The first study should use resettable coding tasks with multiple feasible verification plans and private terminal tests. A second task family should require multi-stage dependencies and legitimate pending work, rather than merely longer sleeps. Snapshot the repository, agent context, artifacts, random state, and job state before intervention. Pair exogenous disturbances where valid, while allowing action-dependent futures to differ.

Two domains, a shared comparison. We propose coding and browser-workflow tracks with separately reported results. Coding tasks require an actual patch and independent tests; browser tasks require observable changes to a resettable site’s state. A web-interface repair remains a coding task, so it cannot by itself establish transfer to browser navigation. Both tracks use the same distinction between equivalent representations and consequential latency changes. Within each domain, compare the same worker and permitted observations under a clock-aware policy and an explicit duration-estimation/replanning controller. Initial development tasks establish tool reliability; held-out repository and site families test generalization.

Admission before scale. An instance enters the confirmatory study only after its baseline, scoring, reset fidelity, and alternative useful plans have been checked. A reduced repository fixture is not interchangeable with the original benchmark environment. Preserve original commits and evaluator versions, group variants by their underlying task, and keep candidate inventory counts separate from executed-task denominators. Tasks for which both temporal regimes favor the same plan are necessary controls. They must not be relabeled as failed adaptation simply because the agent does not switch.

Strong baselines. Supply the same persistent harness, clock, timeout handling, artifact delivery, and observed history to every strong condition. Compare a fixed progress rule, moving-average duration estimates with explicit replanning, a tuned value-of-information controller, and budget-aware RL where reproducible. A no-clock condition is diagnostic, not the principal competitor. Disclose inference, serving, training, and branch-collection costs. Do not compare a GPU worker to a CPU worker as a test of chronoception.

Outcomes and uncertainty. Jointly report equivalent-pair stability and changed-dynamics utility; neither alone suffices. Assess duration distribution calibration and sharpness, deadline violations, task quality, time to usable delivery, and excess abandoned or wasted work. Estimate candidate-set regret with independent evaluation replications after selecting candidates. Bootstrap over independent tasks or repository clusters, not correlated checkpoints. Count invalid and late outputs as outcomes. Use pilot variance to select confirmatory sample size and freeze tolerances before held-out testing.

What would change our view? If clocks plus simple duration estimates and replanning match the learned model on held-out shifts, the case for learned temporal representations weakens. If apparent improvements

disappear under unit conversion or come entirely from extra compute, the transfer claim fails. If forecasts improve but control does not, prediction is insufficient. If outcome-only training already passes the joint tests, the evaluation criterion may remain useful while the proposed auxiliary training is unnecessary.

12 Objections, boundaries, and next evidence

“This is robustness, not a new faculty.” Unit conversion alone is indeed ordinary robustness. Our position is that temporal equivalence, consequential adaptation, and the cost of acting on evidence must be evaluated together. We retain chronoception as an operational organizing concept; calling the same mechanism adaptive control does not change the required evidence.

“Time invariance is already known.” Yes [17]. We do not claim a new symmetry of control theory. Our narrower proposal is to use validated equivalence and consequential changes together to assess learned LLM-agent timing, and to test a corresponding paired training signal. A paper that only restates the symmetry would not establish the proposed empirical contribution.

“A scheduler can do this.” It may. External implementations are legitimate and must be strong baselines. A successful external scheduler supports system-level chronoception; it does not show that the language model acquired an internal clock. Model training needs a demonstrated benefit beyond that baseline, or a clearly measured advantage such as broader task coverage. A system-level gain must not be relabeled as a model-internal capability.

“Stopping early proves temporal blindness.” It does not. Completion, saturation, lost authorization, infrastructure failure, and formatting failure must be distinguished. Similarly, a deadline-respecting policy that repeatedly runs an uninformative test has not demonstrated useful temporal adaptation. User requests for observation coverage require explicit coverage checks rather than a reward for extra actions.

Evidence status and research sequence. The constructive witness in Section 7 is established by exact arithmetic; it supplies no estimate of model prevalence or treatment benefit. We retain the earlier virtual-time microtask probes and short tool rollouts as implementation evidence, with raw failures and model configurations documented in Appendix C. They do not establish the new intervention criterion or an RL benefit. The next sequence is to validate pair construction, execute the joint tests with strong fixed controllers, demonstrate a repeatable generalization gap, and only then train and test the proposed objective. Chronoception remains the main question: whether an agent connects measured time, predicted time, and action in a way that transfers when timing changes.

13 Implications for agent builders and evaluators

Evaluate the deployed system, then localize the capability. Many agents combine a language model with timers, queues, retry policies, and persistent state. If an external scheduler handles adaptation, that is a useful engineering result. Evaluation should first ask whether the complete system serves the user and then identify which component supplies the capability. Removing a scheduler from one condition while retaining it in another confounds timing competence with infrastructure. Claims about the language model itself require an additional controlled comparison.

Keep persistence separate from temporal competence. A durable process can survive a context reset yet make poor choices about what to do next. Conversely, a competent timing decision can be lost when the harness discards a pending job. Long-horizon evaluations should cross these failure modes rather than treating hours of operation as evidence for both. Record dependency depth, context resets, observed timing changes, and delivered milestones separately. A useful stress test asks whether a forecast and the evidence behind it survive a restart, and whether the policy revises that forecast after new observations.

Prefer the simplest mechanism that survives the tests. Start with measured durations and explicit replanning. Add learned prediction when uncertainty, tool heterogeneity, or changing conditions make a simple estimate inadequate. Add policy learning only when a matched comparison shows that selecting actions is the remaining bottleneck. This sequence avoids rewarding complexity for its own sake. A negative result in which a moving-average scheduler matches a learned agent would narrow the role of learning while still informing reliable system design.

Make temporal claims inspectable. A report that an agent is “time aware” should state which interventions it survived, what information it could observe, and what it delivered. A useful release preserves the timing ledger, intervention specification, failures, and baseline controller alongside successful trajectories. Report both when evidence was obtained and when the resulting action was dispatched: those are different decision states. Researchers should distinguish arithmetic mistakes, poor forecasts, ignored forecasts, and inappropriate user objectives. Each suggests a different intervention; a single deadline score cannot identify the appropriate remedy.

14 Evidence boundaries of this release

This preprint advances an evaluation position, not a validated new training algorithm. The new-task factorial screen contains 128 completed episodes; the subsequent slowdown-and-recovery screen attempted all 36 planned conditions. There are 34 valid timed outcomes and two retained conditional-plan schema failures. The original credit interruption and later execution blocks are disclosed. Failure incidence uses all attempted conditions; quality and timing means condition on valid runs and must not be read as intention-to-treat utility. Appendix L reports actual denominators and the interruption history. No timed outcome is imputed for either schema failure. Appendix N adds 32 short three-milestone trajectories with charged probes, deliberate scheduler delay, model replanning, and conditional runtime control. Appendix M adds an offline waiting-cost sensitivity analysis and an exact constructed controller stress test; neither adds live model episodes.

The tasks are short authored workflows with prepared candidate actions, supplied quality increments, and controlled delays. The single-decision screen directly exposes the current service delay. The multistage extension instead uses charged probe durations and a fixed estimator; neither establishes learned duration forecasting. Models share a provider, and repeated conditions are not independent task families. Observed scheduler advantages are descriptive and do not establish general superiority. The experiments demonstrate neither a stable timestamp-prompt benefit nor improvements in autonomous long-horizon work, agent-to-agent coordination, or reinforcement learning. Those stronger claims require held-out task mechanics, uncertain timing evidence, and comparisons with equally informed inexpensive controllers.

15 Conclusion

Useful persistence is the practical objective: retain a delegation when further authorized work serves the user, and deliver or stop when it does not. Time budgets become useful only through decisions about work. We argue that agent chronoception should be evaluated by whether temporal evidence supports those decisions: preserving useful plans under equivalent changes, revising them when their value changes, and choosing when to verify, deliver, continue, or stop. Punctuality, persistence, and fluent duration reports each provide incomplete evidence. The constructed example establishes this distinction. The controlled screens probe budget aging, handoff preservation, and the cost of deliberation, including repeated milestones under a shared root clock; their short authored workflows do not validate long-horizon generalization.

The broader scope of the position is temporal validity across the boundaries of delegated work: useful evidence must remain interpretable after computation, interruption, and handoff. This is a common evaluation requirement, not an empirical claim that one intervention already solves all these settings.

For benchmark designers, the next step is to pair temporal interventions with validated opportunities for useful continuation. For agent builders, it is to separate measured durations, forecasts, and delivered artifacts in the execution record. For learning researchers, it is to test whether a learned controller adds value beyond equally informed schedulers. These recommendations do not require an internal-clock hypothesis or a learned method to win. They require claims about useful long-horizon behavior to survive a comparison that exposes both costly persistence and premature abandonment.

Conflict of interest

The authors declare no conflicts of interest.

Research artifacts and preparation

The local research artifacts include the exact-arithmetic witness, its machine-readable outputs, the prospective intervention protocol, and the earlier pilot’s versioned code and raw traces. The witness uses authored constants and standard-library rational arithmetic; it executes no language model. This preprint releases the manuscript only. A complete public code and data release is not yet available. Some original source files are temporarily inaccessible during archive recovery, so independent end-to-end reproduction has not been reverified for this release. The preserved multistage result summaries pass checksum verification; that check establishes record integrity, not experimental validity or reproducibility. Generative AI tools assisted literature retrieval, drafting, code preparation, and revision. The model calls reported in the experimental appendices are documented in versioned protocols and raw traces. Discovery, exploratory follow-ups, frozen new-task screening with documented continuation, and interrupted attempts are reported separately. No proposed RL result is reported as completed. **License.** Original manuscript text is licensed under Creative Commons Attribution 4.0 International (CC BY 4.0): <https://creativecommons.org/licenses/by/4.0/>. This statement does not assign a license to separately distributed code or third-party material.

References

- [1] Ankur Samanta et al. *Learning What to Investigate Next: Meta-Reasoning for Long-Horizon Research Agents*. 2026. <https://arxiv.org/abs/2610.02525>.
- [2] Yiruo Cheng et al. *When Should Agents Think? Adaptive Reasoning via Cross-Turn Estimation*. 2026. <https://arxiv.org/abs/2610.12061>.
- [3] Krish Goel, Sanskar Pandey, KS Mahadevan, Harsh Kumar, and Vishesh Khadaria. *Chronocept: Instilling a Sense of Time in Machines*. EACL Student Research Workshop, pp. 437–456, 2026. <https://doi.org/10.18653/v1/2026.eacl-srw.32>.
- [4] Michael Ofengenden and Maksym Andriushchenko. *AgentTime: Can Agents Estimate and Control Their Own Runtime?*. 2026. <https://arxiv.org/abs/2610.09944>.
- [5] Yichuan Ma et al. *Timely Machine: Awareness of Time Makes Test-Time Scaling Agentic*. 2026. <https://arxiv.org/abs/2601.16486>.
- [6] Aaron Wang et al. *On the Clock: Towards Punctual and Productive Time-Budgeted AI Agents*. 2026. <https://arxiv.org/abs/2610.10833>.
- [7] Yize Cheng et al. *Your LLM Agents are Temporally Blind: The Misalignment Between Tool Use Decisions and Human Time Perception*. 2025–2026, v3. <https://arxiv.org/abs/2510.23853>.
- [8] Wenyue Hua et al. *Interactive Speculative Planning: Enhance Agent Efficiency through Co-design of System and User Interface*. ICLR, 2025. <https://arxiv.org/abs/2410.00079>.
- [9] Zhengru Fang et al. *Inference-Time Budget Control for LLM Search Agents*. 2026. <https://arxiv.org/abs/2605.05701>.
- [10] Aneesh Muppidi et al. *Finding the Time to Think: Learning Planning Budgets in Real-Time RL*. 2026. <https://arxiv.org/abs/2606.26463>.
- [11] Stuart Russell and Eric Wefald. *Principles of Metareasoning*. Artificial Intelligence, 49:361–395, 1991. [https://doi.org/10.1016/0004-3702\(91\)90015-C](https://doi.org/10.1016/0004-3702(91)90015-C).
- [12] C. Nicolò De Sabbata et al. *Rational Metareasoning for Large Language Models*. 2024–2025, v3. <https://arxiv.org/abs/2410.05563>.
- [13] Cuong Dang, Hoang Anh Just, and Ruoxi Jia. *Selecting Long-Horizon Trajectories for Reliable and Efficient Terminal-Agent Training*. 2026. <https://arxiv.org/abs/2610.05831>.

- [14] Lirui Luo et al. *Coding Agent Memory Post-training: Unlocking the Memory Potential of Pre-trained File Operations for Long-Horizon Tasks via Reinforcement Learning*. 2026. <https://arxiv.org/abs/2609.34422>.
- [15] Richard S. Sutton, Doina Precup, and Satinder Singh. *Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning*. Artificial Intelligence, 112:181–211, 1999. [https://doi.org/10.1016/S0004-3702\(99\)00052-1](https://doi.org/10.1016/S0004-3702(99)00052-1).
- [16] Anthropic. *What’s new in Claude Sonnet 5.5*. API documentation, accessed October 12, 2026. <https://platform.claude.com/docs/en/models/sonnet-5-5/whats-new-sonnet-5-5>.
- [17] Zoran Tiganj, Samuel J. Gershman, Per B. Sederberg, and Marc W. Howard. *Estimating Scale-Invariant Future in Continuous Time*. Neural Computation, 31(4):681–709, 2019. https://doi.org/10.1162/neco_a_01171.
- [18] Romain Froger et al. *Gaia2: Benchmarking LLM Agents on Dynamic and Asynchronous Environments*. ICLR, 2026. <https://arxiv.org/abs/2602.11964>.
- [19] Sudarshan Sreeram, Young D. Kwon, and Cecilia Mascolo. *Tempora: Characterising the Time-Contingent Utility of Online Test-Time Adaptation*. 2026. <https://arxiv.org/abs/2602.06136>.
- [20] P. Berry, K. Conley, M. Gervasio, B. Peintner, T. Uribe, and N. Yorke-Smith. *Deploying a Personalized Time Management Agent*. AAMAS Industrial Track, pp. 1564–1571, 2006. <https://www.sri.com/publication/artificial-intelligence-pubs/deploying-a-personalized-time-management-agent/>.
- [21] Eric Loreaux. *Multi-agent Teams under Latency Pressure and Budgets*. Anthropic Claude Cookbook, September 18, 2026. Engineering example, not a peer-reviewed benchmark. <https://platform.claude.com/cookbook/patterns-agents-latency-multi-agent>.
- [22] Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. *Darwin Gödel Machine: Open-Ended Evolution of Self-Improving Agents*. 2025–2026, v3. <https://arxiv.org/abs/2505.22954>.
- [23] Rong He. *Does Failure Feedback Direct Agent Self-Modification?* 2026. <https://arxiv.org/abs/2610.14662>.
- [24] Frederick Callaway, Sayan Gul, Paul M. Krueger, Thomas L. Griffiths, and Falk Lieder. *Learning to Select Computations*. UAI, 2018. <https://arxiv.org/abs/1711.06892>.
- [25] Paras Dahal et al. *Thinking Before Thinking: Scaling Agentic Inference Through Meta-Reasoning*. 2026. <https://arxiv.org/abs/2609.38147>.
- [26] gRPC Authors. *Deadlines*. Documentation, accessed October 15, 2026. <https://grpc.io/docs/guides/deadlines/>.
- [27] Kazuki Nakayashiki. *When Stale Constraints Go Unchecked: Budgeted Verification Failures in Inherited Agent Memory*. 2026. <https://arxiv.org/abs/2608.25553>.
- [28] Amihay Elboher, Ava Bensoussan, Erez Karpas, Wheeler Ruml, Shahaf S. Shperberg, and Solomon E. Shimony. *A Formal Metareasoning Model of Concurrent Planning and Execution*. 2023. <https://arxiv.org/abs/2303.02664>.
- [29] Yoonchang Sung, Shahaf S. Shperberg, Qi Wang, and Peter Stone. *Effort Allocation for Deadline-Aware Task and Motion Planning: A Metareasoning Approach*. 2024. <https://arxiv.org/abs/2410.05828>.

A Formal definitions and constructive argument

These definitions support the conceptual argument. They are not new empirical results.

Elapsed-time accounting For serial execution, an audit ledger can satisfy

$$T_{\text{wall}} = \sum_{k=1}^K (\Delta_k^{\text{infer}} + \Delta_k^{\text{tool}} + \Delta_k^{\text{other}}). \quad (1)$$

The intervals must be disjoint and exhaustive. Concurrent jobs require an event graph and critical-path accounting; summing overlapping latencies overstates elapsed time. Multiplying action count by an average duration is a descriptive approximation, not a general law or evidence that a policy has learned time.

Observation model and equivalence conditions Let $x = (h, \mathcal{C}, P_\Delta)$ describe a task history, a declared user contract, and the environment’s duration dynamics. The policy receives only $z = O(x)$, the permitted observation, rather than the hidden distribution P_Δ . A contract $\mathcal{C} = (D, B, \mathcal{D}, \mathcal{E}, u)$ specifies deadline, resource cap, delivery rules, stopping conditions, and quality–delay preferences.

Representation equivalence. A transformation g changes units or a clock origin while preserving the same physical deadlines and durations. All affected timestamps, timeout arguments, and unit declarations must change consistently. After mapping outputs into a common action space, the policy distributions should approximately agree:

$$\pi(\cdot \mid O(x)) \approx g_{\#}^{-1} \pi(\cdot \mid O(gx)). \quad (2)$$

Here $g_{\#}^{-1}$ converts action parameters back to canonical units. We compare task-level decisions, not exact token sequences. Different valid patches or near-tied actions need not match literally. This test measures representation robustness; by itself it does not establish temporal reasoning.

Physical time rescaling is a stronger, conditional control. If every event duration and the horizon are multiplied by α , the decision problem is equivalent only when other time-dependent quantities transform appropriately. For a waiting penalty $\int_0^H w(t) dt$, use $w'(t') = w(t'/\alpha)/\alpha$. Continuous discount rates, expiry events, inference latency, and duration-based charges also require corresponding transformations. Fixed per-action costs may remain fixed. Under an action-preserving trajectory correspondence with unchanged transitions and total returns, policy values are preserved and optimal action sets correspond. This follows by mapping trajectories and substituting $t' = \alpha t$ in the cost integral; it is not a new invariance theorem. A simulator can enforce these assumptions. A real system in which only tool sleep is scaled generally cannot.

Joint diagnostics

The corresponding joint report. For a finite canonical plan set $\mathcal{O}(x)$, report equivalent-pair total variation

$$E_{\text{eq}} = \mathbb{E}_{(x, gx)} \left[\frac{1}{2} \sum_{o \in \mathcal{O}(x)} \left| \pi(o \mid O(x)) - g_{\#}^{-1} \pi(o \mid O(gx)) \right| \right] \quad (3)$$

and state-conditional candidate regret

$$R(x) = \max_{o \in \mathcal{O}(x)} \mathbb{E}[G \mid x, o] - \mathbb{E}_{o \sim \pi(\cdot \mid O(x))} \mathbb{E}[G \mid x, o]. \quad (4)$$

The first detects irrelevant sensitivity; the second detects costly insensitivity or misadaptation. Estimate R separately on preference-reversing, preference-preserving, and uncertain pairs, using independent outcome samples. A known-law comparator may have more information than the policy: disclose that gap rather than interpreting all regret as a capability failure. Exact sequence agreement is inappropriate for open-ended actions; use prespecified semantic option categories and report categorization uncertainty. Low total variation is a diagnostic expectation, not a penalty for choosing another equally useful plan. Paired utility and near-optimal action-set membership should accompany it. Do not collapse these measurements into an arbitrary weighted “chronoception score.”

Delivered utility and continuation value Let H be the fixed deadline or an independently sampled inspection time specified before execution. For the artifact Y_t actually delivered by time t , use the declared objective

$$J(\pi; \mathcal{C}) = \mathbb{E}_{\pi} \left[q_u(Y_H) - \int_0^H w_u(t, Y_t) dt - \lambda_c C_{0:H} - \lambda_a N_{0:H} \right]. \quad (5)$$

C is charged computation and N costly user interruptions. The actual delivered checkpoint, not a private evaluator’s best retrospective selection, determines Y_t . Synthetic preferences support controlled tests but do not measure human satisfaction. Hard deadlines and cancellation remain enforced by the harness.

For a feasible multi-action plan o , define the established continuation comparison

$$A_{\text{cont}}^{\pi}(h, o; \mathcal{C}) = Q^{\pi}(h, o; \mathcal{C}) - Q^{\pi}(h, \text{END}; \mathcal{C}). \quad (6)$$

The plan may include testing, delivery, or waiting for a job. This quantity translates timing into task consequences. A negative estimate for one plan does not prove every continuation is useless.

Prospective learning objective A candidate objective separates three signals:

$$\mathcal{L}(\theta, \phi) = -J(\pi_\theta) + \lambda_{\text{dur}} \mathcal{L}_{\text{proper}}(p_\phi) + \lambda_{\text{eq}} \mathbb{E}_{(x, gx) \in \mathcal{P}_{\text{eq}}} d\left(\pi_\theta(\cdot \mid O(x)), g_{\#}^{-1} \pi_\theta(\cdot \mid O(gx))\right). \quad (7)$$

Use a proper duration score with survival likelihood for censored observations. The last term is a divergence between canonical option distributions on *validated equivalent pairs only*. It must not force agreement after consequential changes. Equivalent paired policies can both be wrong, so task utility is indispensable. For API models without accessible probabilities, evaluate sampled action distributions rather than pretending to train this objective through the endpoint.

Exact separation in the constructed example

Proposition: punctuality and equivalence do not identify adaptation. In this environment, a policy that always chooses the full plan is perfectly punctual and can be perfectly invariant to time-unit changes. Nevertheless, it has strictly positive mean regret over the two equally weighted regimes. More generally, any policy whose full-plan probability p is independent of the regime has mean candidate-set regret

$$R(p) = \frac{1}{2} [0.06(1 - p) + 0.04p] = 0.03 - 0.01p \geq 0.02. \quad (8)$$

Proof. In the fast regime, choosing the targeted plan loses $0.82 - 0.76 = 0.06$. In the slow regime, choosing the full plan loses $0.76 - 0.72 = 0.04$. Averaging the two losses yields the expression. An informed selector choosing full when fast and targeted when slow has zero candidate-set regret. Both it and the constant policy can ignore irrelevant unit changes. Thus, punctuality and representation invariance, even jointly, do not determine whether the policy adapts usefully. $\square$

B Supporting delivery and continuation audit

The following mechanisms support downstream utility measurement. They do not replace the equivalence and adaptation tests in Section 6.

A state record. Store `task_id`, `contract_version`, `deadline`, `budget_remaining`, `elapsed_wall`, `charged_compute`, `delivered_artifact`, `open_hypotheses`, `pending_jobs`, `verification_results`, and `termination_reason`. Record observed facts separately from estimates. The estimated duration of a tool call and the actual measured duration are different fields. Durability must include artifacts and running-job identifiers, not only a natural-language summary of intent.

A small controller before costly end-to-end RL. A first prototype can use features such as remaining budget, last verified result, unresolved failures, recent distinct attempts, active-job status, and duration forecasts. Train a low-capacity predictor of paired branch utility and compare it to a threshold heuristic. Freezing the worker isolates controller behavior and makes branch collection cheaper. Only escalate to full model RL after demonstrating that branch values are learnable and transferable; otherwise additional training may optimize a noisy target.

Termination taxonomy. Log `complete`, `saturated`, `blocked`, `budget_exhausted`, `cancelled`, `infrastructure_failure`, and `voluntary_abandonment` separately. Saturation is a claim requiring evidence, not an automatic pass. A normal language-model finish reason is not a task-level completion certificate. Requiring an exhaustive proof of optimality would itself create endless work, so saturation should be judged relative to an agreed search scope and evidence standard.

Initial pilot design. Use a small, fixed collection of development tasks to debug reset fidelity and metrics, then freeze the protocol before a held-out study. Cross foreground versus background contracts with tight versus generous budgets and fast versus delayed tools. Include intermediate delivery as a randomized intervention. Determine the confirmatory task count from pilot variance and a prespecified practically meaningful paired effect, rather than selecting a convenient number of rollouts and calling it statistically sufficient. Separate task families, repository histories, and latency distributions between training and test.

Branch estimator. For a checkpoint h and a prespecified continuation option o , run M paired replications and form

$$\hat{A}(h, o; \mathcal{C}) = \frac{1}{M} \sum_{j=1}^M [G_j(h, o; \mathcal{C}) - G_j(h, \text{END}; \mathcal{C})]. \quad (9)$$

Match exogenous random seeds where valid, but do not force action-dependent events to be identical. Use independent evaluation replications after selecting the best candidate to limit selection optimism. A negative estimate for one option is not evidence that every continuation is useless. Record the candidate set and search effort used to construct the comparator.

Metrics with explicit denominators. For voluntary-end checkpoints sampled from held-out tasks, report the fraction with a positively valued tested continuation, together with uncertainty. For continued-work checkpoints, report the fraction whose selected continuation underperforms ending or immediate delivery. These conditional rates diagnose behavior but do not replace full-episode utility: a policy can change which states it visits. Therefore evaluate both on common saved states and on-policy episodes. Deadline violations include invalid or missing artifacts at D ; do not condition the primary quality report on valid on-time completions only.

Checkpoint selection. The online policy selects which artifact to deliver using permitted development evidence. Private test scores evaluate that delivered artifact afterward. They never select the best checkpoint for the policy. Keep failed attempts in the trace, report artifact regressions, and preserve a known usable result where the task permits rollback. User-facing updates should name a completed change, a verified finding, or a genuine blocker; empty reassurance is not progress.

Long-horizon escalation. Extend from short controlled tasks to multi-stage dependencies and then multi-session work. Keep reset count, dependency depth, and elapsed duration as separate axes. Artificial sleep adds elapsed time but does not create a long-horizon reasoning task. Conversely, memory failures can dominate long tasks even when clocks are perfect. A complete paper must show which axis an intervention improves and where it fails.

C Executed feasibility study and rollout audit

Purpose and evidence boundary. On October 12, 2026, we executed a small controller feasibility study before committing resources to repository-scale collection. It tests action dispatch, independent artifact scoring, contract pairing, and separation of delivery from termination. Four authored normalization states represent an available formatting repair, a two-stage repair, completed supported formats, and an unavailable write permission. These are four states of one narrow task family, not four independent benchmark domains. The two-stage case exercises a multi-action option; it is not evidence of long-horizon credit assignment.

Executable artifacts, controlled time. The artifact normalizes numeric strings with currency symbols, separators, and accounting-style negatives. Eight evaluation fixtures cover these formats and invalid values; their outputs and option rankings are withheld from prompts. Development examples provide online feedback. Candidate repairs are fixed, allowlisted transformations, not model-written code. We cross each state with foreground/background waiting rates and two option durations, producing 16 matched decision cases per controller. The clock is virtual: duration interventions execute instantly and are charged in the ledger. Actual inference wall time is logged separately and excluded from the virtual objective. These results therefore cannot establish deployment latency or wall-clock efficiency.

A transparent scoring model. With terminal inspection, let q_f be the correctness fraction of the last delivered artifact and $q_d(t)$ that of the artifact already available to the user, or zero before delivery. The reduced utility is

$$U = q_f - c \int_0^T [1 - q_d(t)] dt - 0.005T - 0.04N_{\text{extra}}. \quad (10)$$

The foreground and background rates are $c = 0.25$ and 0.005 per virtual unit. These are declared synthetic preferences, not estimates of human patience. In the decision probe, each continuation lasts either two or eight

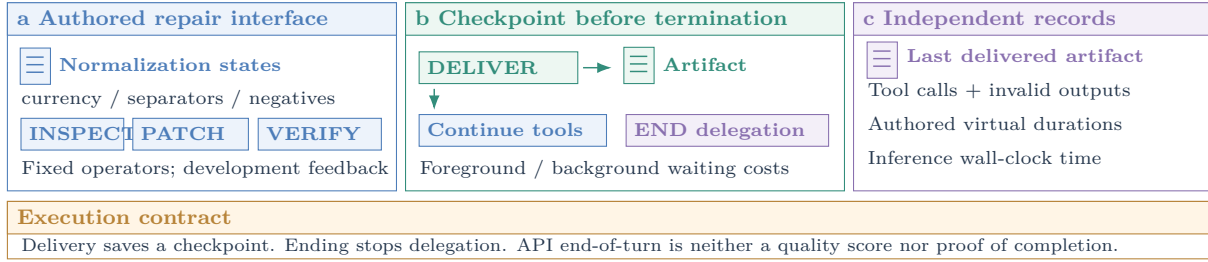


Figure 3: Implemented exploratory normalization interface. Fixed repair operators act on authored states. DELIVER records a checkpoint; END terminates delegation. The diagram is an interface map, not a recorded trajectory. Synthetic waiting costs distinguish foreground and background contracts; virtual task duration and inference wall time are recorded separately.

units under a common twelve-unit cap. The three choices are ending with the current artifact, continuing before delivery, and delivering now followed by continuation. All three outcomes are executed from copies of the same state. Their deterministic utility maximum is an *ex post candidate-set comparator*: it sees evaluation outcomes that the policy does not. A gap can reflect uncertainty about fixture composition as well as poor control; it is not a pure measure of time perception.

Controller / control	Valid / all	Best option / all	Gap, valid only
Qwen2.5-3B-Instruct, cached CPU	15/16	1/16	0.3900
Claude Sonnet 5.5, minimal upfront thinking	16/16	12/16	0.0934
Scripted repair-or-end control	16/16	9/16	0.0981
Outcome-informed comparator	16/16	16/16	0

Table 4: Descriptive decision-probe results on authored development cases, one call per case. “Best option” includes invalid responses as failures in the denominator. The utility gap averages only valid responses; it must be read with the invalid count. Different inference and formatting settings preclude a controlled model ranking. The scripted control knows the authored state category and is an implementation reference, not the strong baseline required to establish a learning contribution.

Configuration and unsuccessful attempts. Qwen2.5-3B-Instruct ran from a pinned cached snapshot on Jetstream CPU, using float32, greedy generation, eight CPU threads, and a sixteen-token output cap; the occupied A100 was not used. One response described an option instead of emitting its identifier. The frontier controller used the API-returned model identifier `claude-sonnet-5-5`, a 256-token cap, low effort, and the documented `between_tools` mode, with no sampling-parameter override [16]. A first request using earlier-model parameters failed. An intermediate default-thinking run produced seven truncated, actionless outputs in sixteen cases. A Sonnet 4.6 development run with a 32-token cap produced four truncated outputs. These attempts are retained separately, not pooled or silently discarded. The final protocol was adjusted after observing these failures; all results are exploratory. Source hashes, raw responses, returned model identifiers, API usage, and measured inference durations are retained. No RL was performed.

Multi-turn delivery test. We also executed eight Sonnet 5.5 episodes: four initial states crossed with the two contracts, at one virtual latency setting and at most eight decisions each. The available tools inspect examples, apply either of two fixed repairs, verify development cases, deliver an artifact, or end delegation. A model’s end-of-turn response dispatches one action; only an explicit end action terminates voluntarily. Each delivered artifact is saved separately, and later private evaluation never selects a better undelivered checkpoint. The harness refuses operations that would exceed the original deadline. Ten automated checks cover fork isolation, contract pairing, invalid-action handling, permission checks, deadline preservation, and delivery without termination.

What this pilot changes. The logs establish that the measurement pipeline can distinguish identical final quality from different delivery utility. They do not show that a learned critic is necessary: the scripted control already performs competitively. The small open model’s invalid formatting and failure on explicit blockage further

Controller	Foreground, four episodes		Background, four episodes	
	Final quality	Utility	Final quality	Utility
Sonnet 5.5	0.875	0.1574	0.875	0.8475
Scripted early-delivery control	0.875	0.5173	0.875	0.8358

Table 5: Integration rollouts under Eq. 10. Both controllers attain the same mean final correctness. Differences arise from the executed sequence and when an artifact becomes available. The scripted policy uses authored state categories and has not been tuned as a general baseline. These eight episodes do not estimate population effects.

Same initial artifact; identical final correctness; different delivery cost

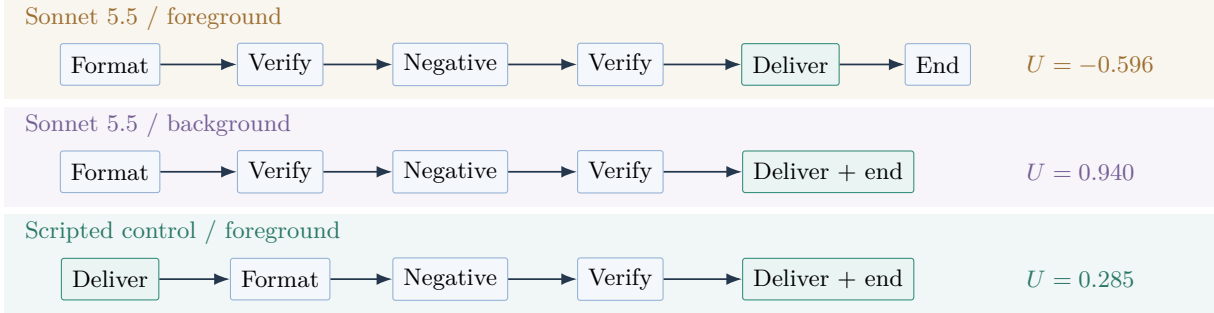


Figure 4: Recorded tool sequences for the repairable microtask; spacing shows order, not proportional time. All three terminal artifacts score 1.0. The foreground Sonnet run first makes an artifact available after both repairs and verifications. Early checkpoint delivery lowers synthetic waiting cost. Immediate ending would deliver the initial artifact at quality and utility 0.5, also exceeding the displayed foreground sequences. The illustration identifies an audit question; it does not establish that every verification was unnecessary or that human users share this utility function.

caution against interpreting every control error as a timing deficit. Expanding this task set or repeating the same prompts would not by itself supply the missing evidence. The next study needs independently authored repository tasks, stronger equally informed baselines, repeated policy samples, and real measured tool durations.

D Resource-aware escalation to counterfactual chronoception

The executed Jetstream pilot used cached weights and CPU resources because existing GPU jobs occupied the A100. A bounded Delta GPU replication launcher has been prepared but has not been submitted: authentication has succeeded, while allocation and isolated serving still require verification. Frontier API runs were launched separately. Neither a prepared scheduler script nor a connected machine counts as a completed experiment.

Stage	Evidence to collect	Gate before the next stage
Implemented	Paired controller probes; short tool rollouts; raw decisions and artifact hashes	Correct reset, delivery, deadline, and scoring semantics. No claim of long-horizon success.
Intervention pilot	Frozen worker, real task states, unit transformations and selective latency shifts	Valid pair construction; repeatable temporal-control errors beyond interface failures.
Paired learning	Duration calibration, equivalence consistency, and matched utility-only RL	Joint stability and adaptation gains on held-out tasks at matched training and inference budgets.
Long-horizon transfer	Multi-stage dependencies, pending jobs, shift recovery, and context resets	Retained temporal state and improved delivery under unseen duration dynamics.

Table 6: Execution stages and decision gates. The development pilot implements the first row; later appendices add narrow timing-intervention screens. The full joint intervention study, paired learning, and long-horizon transfer remain prospective.

Repository collection protocol. Freeze an open worker and an accessible frontier worker with explicit model/version records. Pair the same worker with prompt-only, persistent-clock, progress-rule, value-model, and learned-controller conditions. Use permitted development tests online; reserve independent tests for final scoring. Sample checkpoints before task completion, after useful partial delivery, at apparent saturation, and while a legitimate job is pending. Branch each checkpoint into ending, a bounded coherent attempt, strategy switching, or delivery followed by work. Use development-only variance to set the confirmatory task count; reserve unseen task families before fitting a critic. Do not claim task-level independence for repeated checkpoints from one repository.

A supporting value-learning target. In addition to the paired temporal objective in Section 9, for each sampled state and contract, retain all candidate returns rather than only a winning trace. Fit $\hat{Q}_\theta(h, \mathcal{C}, o)$ to branch utility with task-level held-out calibration. Train the selector over a fixed worker before modifying worker weights, and compare value regression, pairwise preference supervision, and policy RL under identical branch budgets. Add confidence-aware abstention to a verified heuristic when branch support is weak. Both low-value persistence and high-value abandonment must appear in training. A full-policy RL gain is attributable to continuation learning only if it survives an ablation with identical tools, checkpoint delivery, clock state, and action limits.

Timing and serving controls. On dedicated compute allocations, randomize real tool latency independently of task content and keep model serving load measured. Record queue delay outside an episode separately from inference and tool latency inside it. Hold model precision, context cap, permitted calls, and worker weights fixed across controller comparisons. CPU and GPU runs of the present probe can check implementation portability, but their durations are not a fair temporal-competence leaderboard. Larger model downloads and training are deferred until storage, allocation, and serving isolation are verified.

E Coding and browser integration screen

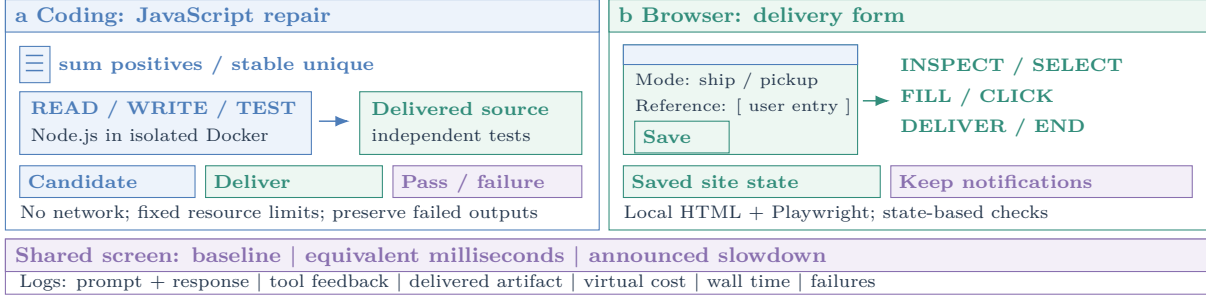


Figure 5: Implemented coding and browser development interfaces. The coding track executes generated JavaScript in an isolated container; the browser track edits a local delivery form and checks saved state. Conditions alter displayed units or authored virtual tool costs. These short screens check integration, not real-time adaptation, public-benchmark performance, or a learning benefit.

On October 13, 2026, we ran a separate development screen comprising two elementary JavaScript repair tasks and two variants of one local browser form. These are authored tasks, not SWE-bench or WebArena instances. Each task was crossed with baseline timing, a coherent seconds-to-milliseconds conversion, and an announced increase in the virtual cost of verification or inspection. There was one episode per cell in a fixed condition order. The browser variants share a site and cannot be counted as independent site families.

The coding worker writes its own JavaScript, optionally executes development checks inside a network-disabled, resource-limited Docker container, and explicitly delivers the source. Independent checks score that delivered source. The browser worker selects, fills, and saves a local form through Playwright; a separate check reads the saved receipt and verifies the required settings and preservation of an unrelated preference. Initial page markup is supplied to the worker, so this does not evaluate screenshot interpretation. Actual tool execution and API latency are recorded, but only declared virtual costs govern the experimental deadline.

In the first model screen, the failed browser episode returned a JSON object with a `tool` key where the parser required `action`; the prompt had not explicitly specified that property name. We therefore attribute the failure

Protocol version	Code	Browser	Interpretation
Scripted preflight	6/6	6/6	Interface checks with known solutions
Initial model screen	6/6	5/6	One ambiguous action-key response
Explicit-schema model screen	6/6	6/6	Repeat after clarifying the interface

Table 7: Successful delivered artifacts over all episodes, separately by development protocol. The two model screens used Claude Sonnet 5.5. Versions are not pooled, and the scripted preflight is not a strong planning baseline.

to an ambiguous interface specification, not temporal incompetence. The second version specified the schema and repeated all cells. Both versions, including the failed response, remain in the raw record. Across them, 24 model episodes required 96 API calls, with 1024 output tokens allowed per call, **between_tools** thinking, and low effort. Source hashes and returned model identifiers accompany the traces.

These runs establish limited integration feasibility. They do not establish useful adaptation: the code tasks permit direct, correct repair without additional tests, and the form tasks can be completed without the inspection action whose cost changes. Unit-paired differences in optional testing cannot support a distributional claim from one episode per cell. The screen motivates a stricter admission criterion for the next study: require independently validated alternative plans, including cases in which a consequential timing change reverses their utility ordering. Longer traces or additional seeds on these same trivial tasks would not fill that evidence gap.

F A measured-delay selection diagnostic

The next development check isolates a narrower question: given explicit estimates of plan quality and duration, does a model select the plan preferred by a declared waiting-cost objective? This is a selection diagnostic, not an autonomous coding or browsing benchmark. Four authored tasks cover numeric-string normalization, stable priority sorting, an invoice form, and a reservation form. Each has a fixed partial plan and a fixed complete plan, evaluated against four declared checks. Both selectors receive development-estimated quality and mean execution duration. Thus this study tests the use of supplied estimates, not learning forecasts or discovering plans.

We crossed each task with four conditions and two repetitions: baseline seconds, equivalent milliseconds with a coherently converted cost coefficient, an additional one-second delay on the complete plan, and an additional sixteen-second delay that reverses the preferred plan. Delays were physically executed. The objective was delivered check fraction minus 0.05 per elapsed second, with a sixty-second delivery limit; these are authored preferences, not estimates of user preferences. Model inference time counts in realized utility. An explicit expected-utility selector receives the same plan estimates. Thirty-two model selections and thirty-two explicit-selector executions were completed in randomized case order, with alternating controller order.

Condition	Strict model execution	Preferred semantic choice
Baseline	8/8	8/8
Equivalent units	6/8	8/8
Ordering preserved	8/8	8/8
Ordering reversed	3/8	5/8

Table 8: Exploratory selection diagnostic. Strict execution counts syntactically accepted selections, not optimal decisions. Semantic choice is a post-hoc reading of the recorded response and cannot replace an executed outcome. Each cell contains only four tasks with two repetitions.

Seven responses contained an appropriate JSON object preceded by explanatory text and were rejected by the original strict parser. Five of these selected the preferred partial plan under the reversing intervention; the other two selected the preferred complete plan under equivalent units. The three accepted reversing responses instead selected the lower-value complete plan. Consequently, low realized utility combines interface rejection and decision error. Treating every failed execution as a chronoceptive deficit would misidentify the mechanism. The explicit selector chose the estimate-preferred plan in all cells, as expected from its construction; this is a reference calculation, not evidence that a new algorithm outperforms agents.

Implications and limits. A prospective interface revision accepts one unambiguous, schema-valid JSON object with optional surrounding text, while rejecting multiple objects or invalid identifiers. Original results remain unchanged, and any rerun must be reported as a separate protocol. The next informative study needs independent task families, stronger model replication, and held-out duration estimates before training a controller. The present diagnostic establishes neither sustained adaptation nor a benefit from reinforcement learning. In particular, accepting incomplete work is part of this artificial graded objective and must not be generalized to tasks with mandatory correctness requirements.

G Bounded controller revision in an authored simulator

We executed a development simulation to screen the prospective methodology in Appendix K. It is not a coding/browser benchmark, a multi-agent rollout, or recursive improvement of an improver. A fixed simulator offers complete or partial work at each stage, under stable, slowing, recovering, and periodically delayed service. All task durations are simulated. A diagnostic observation costs simulated time and reveals the current service multiplier; no-observation controllers update an estimate only after a completed full action. Every trajectory includes a final integration interval.

A Claude Sonnet 5.5 endpoint proposed three bounded parameter revisions, receiving only earlier development scores. The editable fields were update rate, forecast multiplier, diagnostic observation, and integration reserve. The objective and evaluator were fixed. Development, validation, and test used disjoint seed ranges and six, eight, and ten stages, respectively; each split had thirty seeds per regime. Validation selected one candidate before test evaluation. A three-candidate random search provides an equal-candidate-count control, not an equal-compute comparison. Seven fixed policies were also evaluated on the same test cases. Shared authored regimes and new seeds do not establish transfer to independent task families.

The objective averages delivered stage quality and subtracts 0.03 per simulated time unit; late delivery receives zero quality. On the 120 constructed test cases, the selected model revision and selected random candidate both attained mean utility 0.0594, compared with 0.0800 for the always-partial policy. Their delivered quality was higher (0.8863 versus 0.6500), but their additional time erased that advantage. Neither selected candidate missed a deadline. These are descriptive outcomes in an artificial graded contract that permits partial work, not an endorsement of incomplete delivery under mandatory requirements.

The three model proposals took about 5.75 seconds of measured API wall time in this run. That search overhead is reported separately, not converted into simulated task time; the reported policy utility therefore does not establish a positive net value of self-revision. The proposer and evaluator were unchanged across rounds, so the experiment must not be described as full RSI. An earlier diagnostic version mixed per-stage and average quality in its decision rule; its rankings were invalidated, retained locally for audit, and excluded. Four contract checks cover objective arithmetic, observation cost, conservative choice, and rejection of evaluator-changing parameters. No result from the interrupted real-tool expansion is pooled with this simulation.

H Real-tool revision versus delivery

This pilot executes artifact improvement rather than a simulated duration model. Two authored JavaScript tasks and two local browser-wizard variants begin from a common, usable partial checkpoint. The code model writes a replacement module that is executed in a network-disabled, resource-limited container. The browser model returns a bounded sequence of fills, clicks, and checks, executed through two wizard pages and scored from the saved receipt. Each task has one mandatory requirement and three explicitly optional enhancements. These are short, resettable workflows, not public benchmark tasks or autonomous long-horizon trajectories. In particular, improving an authored checkpoint is not recursively modifying the agent’s own controller.

The decision clock starts at the initialized checkpoint. Decision-model inference, revision inference, tool actions, and mandatory verification consume the delivery budget. Initialization is recorded separately. Delivery is timestamped before experiment cleanup; cleanup is separately charged to resource occupancy. A failed or late attempt cannot replace the earlier partial delivery. Quality is the fraction of four checks satisfied, conditional on preserving the mandatory requirement; the initial checkpoint has quality 0.25. Utility subtracts 0.02 per second of waiting, capped at the deadline. Work continuing beyond that deadline remains in the execution record but earns no hindsight credit. These graded preferences are authored and cannot be generalized to tasks whose optional enhancements are mandatory.

The first development protocol ran 48 episodes with five- and thirty-second budgets, comparing immediate delivery, forced revision, and a model gate. Its decision prompt did not quantify the quality increments or provide measured revision-time estimates. We retained it as calibration data rather than interpreting conservative decisions as temporal incompetence. A separate protocol specifies quality increments and supplies domain-level development estimates to both the model gate and an explicit expected-utility gate. The explicit gate uses the observed revision-success fraction and a development 90th-percentile duration, and declines revisions estimated to exceed the remaining budget. These estimates are small-sample forecasts, not guarantees.

The second protocol uses two- and fifteen-second budgets, four task instances, two repetitions, and four arms, totaling 64 episodes. A separately frozen repeat tests the model and explicit gates on 32 further episodes using unchanged prompts, scoring, and calibration. Episode order is randomized within each batch. Stochastic revisions are independent samples rather than identical potential outcomes. Browser variants share wizard mechanics, and both batches reuse the same task families and endpoint. Thus repeated behavior would support a local mechanism hypothesis, not population-level generalization or a training improvement.

Domain	Budget	Initial calibrated batch		Frozen repeat	
		Model gate	Explicit gate	Model gate	Explicit gate
Code	2 s	0/4	0/4	0/4	0/4
Code	15 s	4/4	4/4	4/4	4/4
Browser	2 s	0/4	4/4	0/4	4/4
Browser	15 s	4/4	4/4	4/4	4/4

Table 9: On-time upgrades of an already usable checkpoint, not overall task-success rates. Each cell contains two authored tasks and two repetitions. The repeat reuses the tasks and calibration, so it is not independent task-family replication.

Both batches completed without action-parser errors. Under the two-second browser contract, the explicit gate upgraded all four checkpoints in each batch, whereas the model gate upgraded none. In each batch the model chose revision twice; these attempts finished late. Its other choices retained the partial checkpoint, and one decision call in the repeat itself exceeded the deadline. Under fifteen seconds, both gates upgraded all checkpoints in both domains. The extra model decision typically consumed about one second. This is repeated local evidence of a control-overhead bottleneck, not a causal demonstration that the model lacks temporal representations: revision samples and service times are not identical across arms.

The explicit baseline is also imperfect. At two seconds it declined both code tasks because the domain-level conservative forecast exceeded the budget. In the calibrated batch, forced revision nevertheless upgraded both slug-task checkpoints while both money-task revisions missed the deadline. Aggregating forecasts by domain therefore concealed a useful task-specific distinction. The next comparison should separate forecast granularity, decision computation, and post-decision deadline checks while preserving the same worker and information access. A larger repeat on these same fixtures would not establish the missing generalization.

The resulting methodological lesson is narrower than a new algorithm: assess whether an agent can exploit a feasible improvement after paying for the decision to attempt it, and distinguish the cost of deciding from forecast miscalibration. Metareasoning overhead itself has precedents [24, 25]. The observed pattern motivates controlled tests of stale pre-decision budgets, task-conditioned forecasts, and retained handoff deadlines; it does not establish novelty, RSI improvement, or long-horizon transfer by itself.

I Budget aging and handoff preservation: a controlled screen

This screen distinguishes temporal evidence that has aged from a deadline that was lost. It reuses two authored code tasks and two browser-wizard variants from Appendix H, but fixes the complete candidate and tool sequence in advance. Consequently, it tests admission of prepared work, not independent generation of a repair or autonomous long-horizon planning. JavaScript verification runs in an isolated container; browser updates run through the actual two-step wizard. A declared one-second service delay is injected into each attempted revision. This is a controlled latency intervention, not a measurement of natural tool latency or model thought time.

The snapshot comparison gives both arms the original five-second budget and measured snapshot age, with ages near zero or three seconds. One arm additionally gives the equivalent refreshed remainder. The handoff comparison generates a parent summary under an original five- or eleven-second deadline. Paired children reuse

that same summary, with the parent’s measured inference duration replayed and charged in the second arm; a minimum three-second handoff age is imposed. One child receives only the summary and current elapsed time, and the other also receives the original deadline in a separate field. Decision and prepared-work forecasts are each fixed at 1.5 seconds, explicitly as estimates rather than upper bounds. The parent call, waiting, child decision, injected delay, and tool execution all consume the original budget. Setup and cleanup are recorded separately. Arms have independent API samples and realized service times, so comparisons are not identical potential outcomes.

The initial screen contains 64 episodes: four task instances, two repetitions, and eight conditions per instance. All 32 snapshot decisions follow the supplied feasibility forecasts: revision is selected with a fresh budget and declined after the budget has aged. Explicitly calculating the remainder adds no observed benefit in this small setting. The parent is explicitly instructed that the root clock never resets, making deadline preservation salient rather than measuring unprompted handoff behavior. Manual review of the 16 distinct parent summaries finds the original deadline retained in every summary; this screen therefore provides no evidence of deadline omission during short handoffs.

Two of the 32 handoff episodes nevertheless admit infeasible code revisions, both on the slug task: one with the natural summary and one with the separately restored deadline. At dispatch, about 0.69 and 0.42 seconds remain, respectively; both attempts finish after the original deadline. The summaries also contain an outdated statement that the full five seconds remain, while the child receives a newer elapsed-time observation of roughly three seconds. This is a candidate conflict between old narrative and current timing evidence, not demonstrated loss of the deadline. A shadow post-decision check using the fixed work forecast would veto both attempts. That calculation is a diagnostic baseline, not a randomized demonstration of a remedy.

Across the initial screen, 20 decisions to keep the checkpoint themselves return after the deadline. The partial artifact was already delivered before the decision, so these are late control responses, not 20 newly failed deliveries. One browser parent summary alone consumes roughly ten seconds of an eleven-second budget; both paired children appropriately decline further work. These observations reinforce the distinction between costly decision processes and incorrect temporal decisions. They cannot establish a new failure mechanism from latency alone.

An exploratory follow-up freezes the two tight-budget slug summaries and two corresponding booking summaries from the discovery batch. Each summary is replayed twice per arm. Natural handoff, separately restored deadline, and explicitly refreshed remaining time produce respectively 3/4, 1/4, and 3/4 late revisions on slug, and 0/4, 1/4, and 0/4 on booking. All these late attempts would be rejected by the shadow dispatch-time feasibility check. The selected code failure recurs locally, but the results do not show a reliable advantage for either additional deadline field. Because task and summary selection followed discovery, this is exploratory repetition, not independent confirmation. In particular, simply adding the current remainder does not identify stale-memory reliance as the cause; the model might instead omit its own decision cost or override the supplied work forecast.

A final exploratory diagnostic adds either an explicit sum of the decision and work forecasts or the observation time of the inherited summary, while retaining the refreshed remainder in all arms. On four slug episodes per arm, late revisions occur in 2/4 with only the refreshed remainder, 1/4 with the summed forecast, and 0/4 with the summary timestamp and an explicit freshness rule. These differences are too small and adaptively selected to establish a reliable remedy. The timestamp condition also adds explanatory text, so a matched-content control is needed before attributing an effect specifically to temporal provenance. All attempted fixed candidates pass their task checks; the observed failures are in timely admission, not candidate correctness.

The resulting hypothesis is narrower than deadline propagation: *retaining a temporal constraint may be insufficient when inherited statements about remaining time compete with newer observations, while the act of resolving that conflict consumes further slack.* A confirmatory study should freeze the intervention, use new task families and models, cross timestamp metadata with matched freshness instructions, and retain a cheap dispatch-time guard. It must also include feasible continuations to detect a remedy that merely stops everything. The present screen supports pursuing that question; it does not establish a cross-domain failure rate, a learning improvement, or a novel mechanism distinct from prior stale-constraint and metareasoning research.

J New-task, two-model factorial screen with documented continuation

To test whether the selected discovery pattern transfers, we froze a separate factorial screen before observing its complete outcomes. It uses four new authored task mechanics: merging intervals, parsing CSV records,

saving a catalog shortlist, and updating an issue board. These differ from the earlier slug, numeric-string, and two-step-wizard fixtures. Candidates remain prepared in advance; the agent chooses whether to execute them. Browser actions and containerized code checks are real, but these are short controlled workflows rather than public benchmark tasks or autonomous long-horizon trajectories.

The design crosses four tasks, two models, two budgets, two repetitions, and four intervention arms, yielding 128 completed episodes. A DNS resolution error stopped the initial run after 110 episodes. An explicitly authorized continuation completed the remaining 18 conditions in their original randomized order, without replacing completed outcomes. The original interruption and unresolved API reservation remain in the record. The continuation reuses the exact parent text and its recorded end-elapsed duration; unlike the original in-memory cache, this replay excludes the small unrecorded serialization/write interval. Service conditions also differ between execution blocks. These limitations preclude treating the resumed observations as an uninterrupted replication. Sonnet 5.5 and Haiku 5.5 are models from the same provider, not independent model families. The four arms expose neither addition, a parent-summary observation timestamp, a freshness instruction, or both. All arms already receive the original deadline and current elapsed time. Thus the factorial separates metadata from an instruction to prefer newer timing evidence, rather than comparing information access with no access. Parent summaries and measured parent durations are reused within each matched four-arm group. Both models use the same low-effort setting and a 4096-token output cap; these endpoint settings differ from discovery. The protocol is locally frozen with source hashes, not externally preregistered.

Execution moves to a dedicated temporary worker on Jetstream. API credentials stay on the local controller. Tool time includes SSH transit, actual remote execution, and container cleanup. A one-second service delay is injected and charged; parent waiting and child inference are also charged against five- or eleven-second root budgets. The forecast remains 1.5 seconds each for the child decision and prepared work, not a bound on actual latency. Repeated episodes from four task mechanics are not 128 independent tasks; we report descriptive paired-condition counts rather than a population-level significance claim.

Model	Root budget	Neither	Timestamp	Instruction	Both
Sonnet 5.5	5 s	0/8	0/8	0/8	0/8
Sonnet 5.5	11 s	6/8	7/8	7/8	7/8
Haiku 5.5	5 s	0/8	0/8	0/8	0/8
Haiku 5.5	11 s	8/8	8/8	8/8	8/8

Table 10: On-time upgrades of an already delivered partial checkpoint in the frozen new-task study. Each cell contains four task mechanics and two repetitions, including the documented continuation. At short budgets, retaining the partial checkpoint can be the appropriate action; a low upgrade count is not an overall task-failure rate.

The 128 completed episodes contain 0 revisions infeasible under the forecasts at request time and one actually late revision. In that case, the request was forecast-feasible but the chosen revision would be rejected by the shadow dispatch-time check after inference consumed slack. There are also 37 late control responses choosing to retain the checkpoint; those do not erase the artifact delivered before the decision. These categories must remain separate when interpreting a deadline failure.

Two earlier attempts are retained separately: the local batch stopped on its first tool execution after a Docker timeout; a remote development batch completed nine episodes before a parent response was truncated at its 1024-token output limit. Neither is pooled with this batch. The latter motivated the common output cap and effort setting used here. Infrastructure interruptions and truncated responses remain part of the reproducibility record, not evidence of a temporal-reasoning mechanism.

The new-task comparison is a test of the discovery hypothesis, not an opportunity to select a new winning prompt. Among completed episodes, no revision violates the supplied feasibility test at request time. Consequently, this screen does not reproduce the selected discovery failure across the new task mechanics and does not establish a benefit of timestamp metadata. The late revision instead illustrates a request that expired during inference; a single such observation is not a stable failure-rate estimate or evidence of a novel mechanism. A reliable intervention would need both fewer inappropriate continuations and preserved useful work, consistently across tasks and models. Small differences within this screen cannot establish that criterion, and a null contrast does not prove universal equivalence. No RL training, unrestricted agent self-modification, or long-horizon transfer is demonstrated by this experiment.

K Prospective extension: self-improvement must repay its time

A more demanding extension of chronoception asks when an agent should improve its own controller rather than continue the delegated task. A new forecast model, prompt, memory rule, or scheduling policy can consume the very slack it seeks to use better. Our proposed methodological principle is that *self-improvement is an action inside a temporal contract, not a free outer loop*. This extends the decision problem without assuming that recursive modification is beneficial.

Self-modifying agents are already an established research direction: the Darwin Gödel Machine evaluates agent-code revisions on coding tasks [22]. Recent work further distinguishes repairing a targeted failure from generalization after revision [23]. Consequently, neither a self-editing loop nor a higher selected development score establishes novelty. Our specific proposed test asks whether an improvement survives charging its proposal, evaluation, integration, and rollback time against the original delivery obligation. This is an application of metareasoning, not a claim to invent the value of computation. Earlier computation-selection research explicitly accounts for metareasoning overhead [24]. Recent agentic meta-reasoning also reports that controller overhead can hurt at small budgets [25]. Thus a short-budget penalty from an extra decision call is a diagnostic observation, not a new phenomenon. A stronger contribution would need to isolate failures of temporal-contract preservation or forecast use beyond these established costs.

Let h denote currently available evidence, D the remaining delivery budget, and C the uncertain time consumed by an improvement attempt. Compare continuing with π to proposing and deploying π' through

$$\Delta_{\text{rev}}(h) = \mathbb{E}[U(\pi', D - C) \mid h] - \mathbb{E}[U(\pi, D) \mid h].$$

Here U evaluates delivery on the original user timeline, including elapsed waiting and hard requirements; the loss of C is not charged twice. These values are unknown estimates, not an executable oracle. A revision that improves isolated policy quality may still have negative Δ_{rev} . Conversely, a reusable improvement might repay its cost across future tasks, but the evaluation must declare that deployment horizon instead of silently amortizing search over unlimited future use.

Three methodological commitments. First, preserve the contract: mandatory checks, evaluator access, deadlines, and resource limits are not editable parameters. Second, expose the mechanism: record what changed in forecasting, observation, stopping, or handoff, and compare with a same-budget nonadaptive search. Third, keep improvement and acceptance separate: development failures may guide revisions, validation selects a frozen candidate, and an untouched test set assesses it once. Repeated inspection turns a test set into development data. A claim of recursive self-improvement additionally requires improvement of the revision process itself; tuning a fixed controller with a fixed proposer is only bounded self-revision.

A method portfolio to challenge, not endorse. Useful candidates include moving-average forecasts, rapid change detection, conservative duration estimates, paid diagnostic probes, explicit integration reserves, and an adaptive gate that declines low-value revision. Each has a failure mode: slow updates miss change points, rapid updates chase noise, conservative estimates abandon useful work, probes consume slack, and fixed reserves misallocate it. Compare their combinations against both a simple scheduler and randomized candidate search. The objective is not to collect more named modules, but to identify which mechanism adds value after its costs are charged.

A prospective recursive test. Freeze worker capabilities and the external evaluator. In an initial round, allow a proposer to revise only a controller or its parameterization. In a separate arm, allow it to revise its proposal strategy using development feedback. Cross those arms with true versus matched irrelevant feedback, and compare equal proposal counts as well as equal elapsed and monetary budgets. Evaluate target repair, related-failure transfer, held-out net utility, and regression separately. Keep cancellation and mandatory verification as hard constraints. If a fixed proposer or random search matches the recursive arm, the stronger RSI claim fails; a simpler control method may still be useful.

Present evidence. Appendix G reports a small simulation of bounded parameter revision. It does not show an RSI gain: the selected model proposal ties a random-search candidate and loses utility to a cheap fixed policy under the authored objective. That negative result is useful for screening methodology, not proof about deployed agents. Appendix H adds a narrow real-tool artifact-revision pilot. Broader code, browser, and parent-helper experiments remain necessary before a claim about useful long-horizon self-improvement.

L Observed slowdown and recovery: a paid-controller screen

We planned 36 exploratory episodes crossing two authored task mechanics (interval merging and catalog short-listing), three service regimes, three controllers, and two repetitions. Twenty-eight episodes completed before an HTTP 400 response stopped model calls. A diagnostic continuation returned an explicit insufficient-credit error; two remaining scheduler episodes were then completed without API calls. A later continuation attempted all six remaining model conditions with unchanged prompts and scoring: four yielded valid outcomes and two produced schema-invalid plans. Thus all 36 conditions were attempted, with 34 valid timed outcomes. No timed outcomes are imputed for failures. A partial candidate preserves the mandatory requirement and passes two of four checks; a full candidate passes all four. The previously delivered checkpoint passes one. Both candidate branches were verified before scoring. These are real container and browser operations over prepared candidates, not generated repairs or autonomous multi-stage planning.

The service delay is controlled: unchanged means one second before and after the update; slowdown changes one to twelve seconds; recovery changes twelve to one. A worker observation explicitly reports the current injected delay. This supplies timing evidence, not an estimate learned from sparse traces. Every controller has the same twelve-second root budget, quality objective, initial service state, and observation interface. Full-work forecasts add half a second to the observed delay; partial work is forecast at 0.9 seconds. Actual delays, tools, SSH transit and verification are charged. Forecasts are not upper bounds.

The scheduler chooses the feasible plan with largest supplied forecast utility. The replanner makes one Sonnet 5.5 call after observing the change. The conditional controller makes one Sonnet 5.5 call before observation to specify an ordered action list and a reserve; the runtime evaluates that restricted template against the new state and actual remaining budget. Its generation cost is charged on every episode, not amortized. A one-second wait precedes the observation. Policy generation, model inference, observation, waiting, execution, verification and container cleanup all count from the initialized delivered checkpoint. Worker startup and checkpoint initialization are outside that boundary; setup is recorded. Thus this is not a cold-start deployment-latency comparison. Conditional generation delays its observation, so the comparison includes deployment overhead rather than isolating cognition at identical instants.

Regime	Controller	n	Quality	Utility	Seconds	Late work
Recovery	Conditional	2	1.00	0.925	3.76	0
Recovery	Replan	4	1.00	0.897	5.16	0
Recovery	Scheduler	4	1.00	0.948	2.59	0
Slowdown	Conditional	4	0.50	0.392	5.40	0
Slowdown	Replan	4	0.50	0.403	4.85	0
Slowdown	Scheduler	4	0.50	0.464	1.79	0
Unchanged	Conditional	4	1.00	0.891	5.44	0
Unchanged	Replan	4	1.00	0.901	4.95	0
Unchanged	Scheduler	4	1.00	0.949	2.53	0

Table 11: Descriptive means in the exploratory paid-controller screen. The planned cell contains two task mechanics and two repetitions; actual cell counts are shown. All 36 conditions were attempted; n counts valid timed outcomes only. Two conditional recovery plans failed the schema and are excluded from these means, not counted as successful runs. Utility subtracts 0.02 times elapsed seconds, capped at twelve seconds, from delivered quality. Delays and quality increments are authored. Counts do not establish population-level effects.

In all 34 valid timed episodes, the selected candidate matches the disclosed regime: partial work after slowdown and full work under unchanged or recovered service. No completed work attempt misses its deadline. Within each observed regime, the scheduler has the same mean delivered quality and a shorter mean elapsed time than either model-based controller. Because these means exclude schema failures, task coverage is tiny, and forecasts and quality increments are supplied, these are descriptive observations rather than a general superiority claim. They provide no evidence that model-based conditional planning adds value on these fixtures.

A post-hoc comparison restricted to matching task, regime, and repetition identifiers gives twelve replanner-scheduler pairs and ten conditional-scheduler pairs. Delivered quality is identical within every observed pair. Replanning takes an additional 2.68 seconds on average, and conditional planning an additional 2.81 seconds; the scheduler is faster in all twenty-two valid comparisons. These are descriptive comparisons of independent live executions, not paired potential outcomes under identical service noise. They exclude the two schema failures

and do not justify an independence-based significance claim from two task mechanics.

The conditional controller’s runtime supplies the state-dependent branch selection. Success therefore cannot be attributed solely to a language model’s acquired chronoception. A cheap scheduler has access to the same disclosed quality and duration information; it is a necessary attribution baseline rather than an intentionally weak comparator. The experiment tests whether these supplied interfaces can implement the proposed comparison and exposes the cost of invoking a model. It does not establish learned duration estimation, novel scheduling, long-horizon adaptation, or a stable advantage of conditional planning.

The initial attempt stopped before a scored episode because a response contained an explanatory prefix followed by one valid JSON object. The fresh scored batch reused the earlier studies’ unique-object decoder and retained complete raw responses; prompts, task candidates, and scoring did not change. Both parser versions remain archived. The scored batch later stopped on HTTP 400 before a model response was available; its first error body was not captured. A diagnostic continuation returned an explicit insufficient-credit error before completing another episode. Only two remaining scheduler conditions were then executed in a separate block with unchanged logic. On October 17, the six unattempted model conditions were run in separate continuation blocks. Two conditional recovery plans returned a priority list containing only PARTIAL and KEEP, omitting the required FULL entry; the frozen validator rejected both. Each assertion stopped its batch. A documented continuation then ran only untouched conditions, without retrying either failure, repairing the answer, or changing prompts. This continuation-after-stop is an execution-procedure amendment, not an uninterrupted run. Schema failure incidence is 2/12 for conditional planning (2/4 in recovery), 0/12 for replanning, and 0/12 for the deterministic scheduler. These are descriptive counts, not evidence of a general temporal failure mechanism; the conditional schema is more demanding. Valid-only quality and utility can be optimistic when invalid plans are excluded, and no end-to-end utility is imputed for those failures. No completed record was overwritten, and the unresolved reservation remains in the budget ledger. Service conditions can differ between blocks. No prompts were selected after observing scored outcomes.

M Offline sensitivity and constructed controller stress test

Observed-trace sensitivity. We reanalyzed the 34 valid timed adaptation episodes, without issuing additional model calls. Twelve replanning and ten conditional-planning observations have a matching scheduler observation for task, regime, and repetition. Reusing a scheduler observation across comparisons does not create independent evidence. Every observed quality difference is zero and every model-controller elapsed-time difference is positive (Figure 6, left). Consequently, for the linear objective $U = Q - \lambda t$, $\Delta U = -\lambda \Delta t$: at $\lambda = 0$ every comparison ties, and for every $\lambda > 0$ the scheduler has higher utility in these observations. We checked $\lambda \in \{0, 0.005, 0.01, 0.02, 0.05, 0.1\}$. This is a post-hoc sensitivity analysis of recorded outcomes, not a rerun under changed preferences; a controller exposed to another objective might choose different actions. It supplies no extrapolation beyond the observed pairs or inference about the two schema-invalid conditions.

Exact finite-grid experiment. Independently of those observations, we enumerated $3 \times 4 \times 3 \times 3 \times 3 \times 3 = 972$ authored states using rational arithmetic. Remaining budget $B \in \{2, 4, 8\}$, decision cost $c \in \{0, 0.5, 2, 4\}$, work duration $d \in \{0.5, 2, 6\}$, integration reserve $r \in \{0, 1, 3\}$, incremental quality $g \in \{0, 0.25, 0.5\}$, and waiting penalty $\lambda \in \{0, 0.02, 0.1\}$ are crossed exhaustively. Continuing costs $c + d + r$ and improves value only if it meets B and $g - \lambda(c + d + r) > 0$. Stopping immediately retains the prior artifact with zero incremental utility. Costs and gains are exact and known in this construction; estimating them is not tested.

Five deterministic rules isolate the diagnostic distinctions: STOP always declines; STALE admits when $d + r \leq B$; RESET omits decision and integration costs and admits when $d \leq B$; VETO admits when $c + d + r \leq B$; EXACT additionally requires positive net value. EXACT is a reference rule with stipulated perfect information, not a learned or newly proposed algorithm. Its zero error is true by construction. The rules are prospective at the common checkpoint; a real runtime veto invoked after a paid decision cannot refund the sunk cost of that decision. Our admission diagnostic does not model that refund or assert a utility guarantee for such a runtime implementation.

The STOP rule misses 231 positive-value feasible continuations; STALE and RESET admit 162 and 342 late continuations respectively. VETO eliminates late continuations in this exact setting but still admits 183 feasible continuations with non-positive net value. Counts reflect this grid and its equally weighted states, not failure rates of deployed agents. No confidence intervals or significance tests apply. The experiment makes explicit why both early abandonment and unproductive persistence belong in an evaluation. It establishes neither uncertain-

forecast robustness nor an intervention effect on LLM agents. Source, complete states, observed-input hashes, and summaries are retained in the local robustness artifact directory.

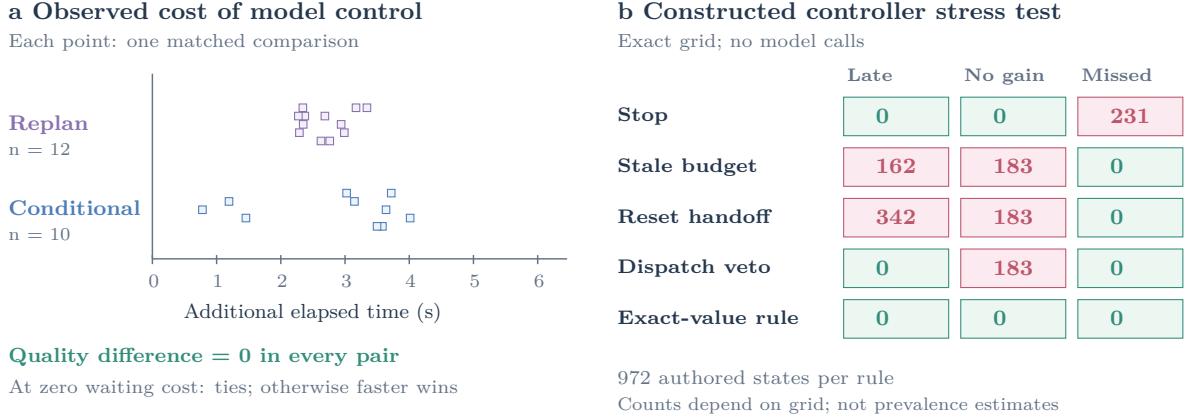


Figure 6: Two separate evidence types. Left: all observed matched controller–scheduler elapsed-time differences; all matched delivered-quality differences are zero. Points are descriptively jittered vertically, not uncertainty estimates. Right: exact counts from a separate authored finite grid. A positive count indicates the named diagnostic event; the perfect-information rule is a reference, not a trained agent. These offline analyses add no model episodes and exclude the two schema-invalid runs from the observed comparison.

N Three milestones under one root clock

Question and design. Does paying for a new decision change which work can still be delivered? We locally froze a separate exploratory study with 32 trajectories: two reused task mechanics, two service regimes, four controllers, and two repetitions, interleaved in a fixed randomized order. Each trajectory contains three repetitions of its prepared fixture, released at root times zero, eight, and sixteen seconds, with cumulative deadlines eight, sixteen, and twenty-four seconds. These are separate milestones under one root clock, not a single unfinished artifact becoming slow and then recovering. A prepared checkpoint scores 0.25 at each milestone; partial and full candidates score 0.5 and 1 after actual verification. Each milestone’s earned quality is fixed at its deadline. The primary reports are mean delivered milestone quality and charged root elapsed time. The authored secondary utility subtracts 0.01 times elapsed seconds, capped at twenty-four, from mean quality.

Timing evidence and controls. The hidden injected latency is 0.15 seconds throughout the constant condition and 0.15, 2.5, and 0.15 seconds across the changing condition. Controllers receive the measured duration of a charged probe, comprising the injection and an actual worker round trip, not the configured latency value. The simple forecast is this measurement plus 0.6 seconds for full work and 0.9 seconds for partial work. This is a noisy, hand-specified estimator, not a learned forecast. The probe does no candidate work. Initialization, probing, waiting for milestone release, model inference, injected execution delay, real code/container or browser operations, verification and cleanup all consume the root clock; persistent worker startup is excluded.

The immediate scheduler maximizes supplied forecast utility among feasible candidates. The delayed scheduler waits 2.7 seconds after each observation before recomputing feasibility; this delay was fixed before new outcomes, by rounding the earlier observed replanner overhead. The replanner makes one model call after each observation. The conditional controller generates one priority/reserve plan before the trajectory; the runtime selects a currently feasible action at each milestone without another model call. All model calls use the same Sonnet 5.5 endpoint and output cap as the previous screen. Unlike that earlier schema, this new protocol permits nonempty priority subsets; it does not repair or relabel the earlier invalid outputs. Conditional generation is charged once to the root clock, not amortized across unrelated trajectories. Different realized observation times and decision delays make these deployment comparisons, not isolated tests of reasoning ability.

In the changing regime, mean delivered quality is 1.000 for the immediate scheduler, 0.833 for the delayed scheduler, 0.750 for replanning, and 1.000 for conditional control. The replanner records 4 late milestone attempts across its four changing-regime trajectories; the delayed scheduler records 0. Figure 7 exposes where observation,

Regime	Controller	n	Quality	Seconds	Late work	Invalid
Constant	Scheduler	4	1.000	16.76	0	0
Constant	Delayed scheduler	4	1.000	19.47	0	0
Constant	Replan	4	1.000	17.87	0	0
Constant	Conditional	4	1.000	16.78	0	0
Slow/recover	Scheduler	4	1.000	16.75	0	0
Slow/recover	Delayed scheduler	4	0.833	19.45	0	0
Slow/recover	Replan	4	0.750	20.40	4	0
Slow/recover	Conditional	4	1.000	16.77	0	0

Table 12: All 32 multistage trajectories, grouped descriptively. Each row has two task mechanics and two repetitions. Late work counts milestone attempts, while invalid counts trajectories. Separate stage outcomes and cumulative deadlines are retained in the raw record. Repeated milestones do not constitute independent task families.

deliberation and work consume the same allowance. These observations support reporting feasible work at dispatch, rather than treating the observation-time budget as an enduring entitlement. They do not show a model advantage over the immediate scheduler, and two repetitions of two fixtures cannot establish a population-level failure rate.

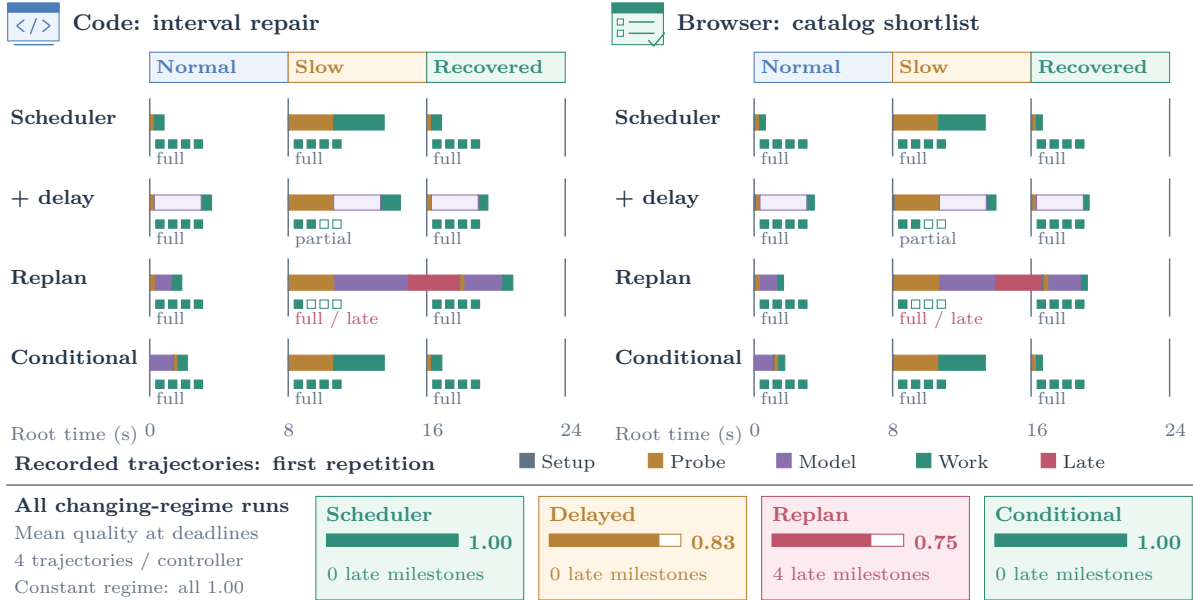


Figure 7: Measured time costs and delivered quality. Top: the predeclared first repetition of each changing-regime task, on a common root clock. Colors identify charged setup, probes, model calls and work; red work finishes late. Outlined purple segments are deliberate delays; gaps wait for milestone release. Stage labels name the executed action, and four squares encode deadline-fixed quality in quarter units, not independent trials. Bottom: all changing-regime trajectories (four per controller); all constant-regime means are 1.00. Conditional control matches, but does not improve on, the immediate scheduler.

Attribution and limits. The fixed scheduler delay does not exactly match each stochastic model call. Its stale suggestion and fresh choice are logged, but the stale suggestion is unexecuted. Likewise, the model’s dispatch-time forecast veto is logged but not enforced; neither shadow diagnostic establishes a causal guard benefit. Failures may reflect action choice, uncertain execution cost, or deliberation time, which these comparisons do not isolate. The immediate scheduler and conditional runtime share a forecast interface, so conditional success cannot be attributed solely to learned model capability. Short repetitions of prepared fixtures do not establish independent task transfer, autonomous coding, personal-agent deployment, reinforcement-learning gains, or natural human waiting preferences.