Evaluating Governed LLM-Driven ML Experimentation: Lifecycle, Provenance, and Failure Metrics

Dowling Wong
dowling.wong@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Abstract

Autonomous ML experimentation agents can report task improvements without showing whether their actions were bounded, auditable, or reproducible. We present a governed harness for LLM-driven ML experimentation in which a control plane, not the manager, owns trial lifecycle, edit-scope validation, keep/discard authority, and an append-only ledger. The ledger records proposal, patch reference, run log, parsed metric, decision, provenance IDs, and failure taxonomy for every trial, making governance properties measurable alongside task metrics. We evaluate the same contract across six ML nodes: a ResNet scientific node with a large-scale replication, a synthetic logistic-regression node, two public OpenML tabular nodes, an MLAgentBench vectorization adapter, and a real LM-training node. Provenance completeness is a structural invariant of the control plane — it holds by construction — and the 1,445 governed trials serve as its empirical certificate: 100% completeness on every node, including high-failure campaigns and protocol-contamination cases detected through reproducibility hashes. Matched ungoverned counterfactuals quantify what the invariant buys: 86% of ungoverned trials leave no ledger record. Memory ablations are used only as diagnostic probes: they reveal node-specific behaviour in proposal repetition, invalid outcomes, and worker failure, rather than establishing a general memory-performance claim. The contribution is a method for treating governance metrics – provenance completeness, lifecycle classification, scope enforcement, and failure taxonomy coverage – as evaluation criteria for autonomous ML experimentation systems.

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**; • **Software and its engineering** → *Software testing and debugging*; • **General and reference** → *Evaluation.*

Keywords

agentic AI evaluation, autonomous ML experimentation, governed harness, control plane, append-only ledger, memory ablation, failure taxonomy, reproducibility, bounded execution

ACM Reference Format:
Dowling Wong. 2026. Evaluating Governed LLM-Driven ML Experimentation: Lifecycle, Provenance, and Failure Metrics. In *Proceedings of KDD Workshop on Evaluation and Trustworthiness of Agentic AI (KDD ETAAI'26).* ACM, New York, NY, USA, 21 pages.

KDD ETAAI'26, Jeju Island, Republic of Korea
2026.

1 Introduction

Autonomous ML experimentation needs evaluation criteria for the experimental process, not only the final task score. A higher validation AUC does not reveal whether the agent edited only allowed files, whether failed trials were preserved, whether no-op patches were rejected, whether keep/discard decisions were made by an external verifier, or whether the system repeatedly attempted known-bad ideas. Without explicit governance metrics, none of those properties are observable — and for scientific ML, observability of process determines whether an autonomous experiment has *auditability*: a traceable, reproducible record of every decision and its evidence.

This paper proposes a governance-metric evaluation protocol for autonomous ML experimentation and a harness that instantiates it. The central question is not only *did the metric improve?* but *did the agent behave in a bounded, auditable, and reproducible way?* In the Agent = Model + Harness decomposition [15], manager and harness together form a compound AI system [19]: the model proposes actions while the harness supplies execution boundaries, state tracking, and verification. We instantiate this as a governed control plane: a manager proposes one bounded change, a worker materialises the patch and run artifacts, and the control plane owns lifecycle, scope validation, metric parsing, keep/discard decisions, memory updates, and append-only records — none delegated back to the agent.

The protocol reports six governance metrics alongside the task metric: acceptance rate (AR), invalid-action rate (IR), repeated-bad rate (RBR), provenance completeness, artifact-reference completeness, and a failure taxonomy. These make process behaviours inspectable in the audit record: invalid edit attempts, missing metrics, repeated degraded proposals, no-op patches, stale pending trials, and unauditable decisions.

We validate the protocol across six ML nodes and 1,445 governed trials. Stated up front: 100% provenance completeness is not an empirical discovery but a structural invariant — it holds by construction of the control plane (Section 3.4). What the trials contribute empirically is certification of the invariant across heterogeneous task types, worker classes, and failure regimes: the same contract transfers to six nodes unchanged, surfaces total worker failure and bounded-proposal exhaustion, detects protocol contamination through reproducibility hashes, and classifies every outcome explicitly. A matched ungoverned counterfactual quantifies what governance buys: without the control plane, 43 of 50 trials (86%) leave no ledger record, and a total-failure campaign is indistinguishable from one that never ran (Section 5.3). Memory ablation serves as a diagnostic governance probe: summary context reduces repeated-bad rate on ResNet-trigger but does not transfer

to the synthetic node, confirming that this response is node- and interface-specific rather than a general property of memory format.

Four contributions follow:

- governance metrics as first-class evaluation criteria for autonomous ML experimentation — acceptance, invalid-action, repeated-bad, provenance completeness, artifact-reference coverage, and failure taxonomy — assigned by the control plane and validated structurally (Section 4.5);
- a governed control-plane protocol separating proposal, execution, validation, and keep/discard authority, with the manager as a replaceable component;
- an append-only audit schema recording proposal, patch, run log, parsed metric, decision, and failure category for every trial, validated across private, synthetic, public OpenML, and MLAgentBench-adapter nodes, three memory modes, and dedicated stress campaigns — together with a matched ungoverned counterfactual in which 86% of trials leave no ledger record;
- a real LM training stress case (`autoresearch_linux`) demonstrating governance integrity under high failure rates: the no-memory arm produces 100% `runtime_error` failures on this specific task and interface configuration, yet all 240 trials are classified with complete provenance, showing that control-plane governance is independent of the cause of failure.

2 Background and Related Work

The key distinction of this paper is that we evaluate the *harness* that governs the agent loop—lifecycle states, scope enforcement, provenance, and failure classification—not only the model that proposes edits.

ML-agent benchmarks and code-space exploration. AIDE [10] is the closest academic system: both use LLMs to propose atomic code changes evaluated by a hard-coded metric, but they differ fundamentally in where lifecycle authority lives. AIDE is a coupled optimisation loop — search policy, evaluator, and solution summarisation share state inside a single process — and uses in-memory tree nodes as its solution store. There is no append-only external ledger, no pending-trial guard, and no typed failure taxonomy; if the process crashes, in-progress solutions are unrecoverable and mid-trial divergence is invisible. AIDE has no editable-path whitelist: any code change can be proposed without scope enforcement. The present work inverts this design: the control-plane contract is fixed and externally visible; the manager and worker are replaceable without altering governance; every outcome—whether kept, discarded, or crashed—produces an immutable typed audit record before the next trial begins. The editable-path whitelist is a precondition guard, not a post-hoc filter.

MLAgentBench [8], MLE-bench [4], FML-bench [20], and MLGym [12] address a different evaluation target: they ask *did the agent improve performance on this task?* This paper asks *did the harness guarantee that every trial was bounded, classified, and provenance-tracked, regardless of outcome?* The two questions are complementary rather than competing. This paper wraps an MLAgentBench-style task (NumPy vectorization speed) as the `mlagentbench_vec` node, demonstrating that any task expressible as a NodeSpec receives the same governance coverage, but the governance evaluation is independent of task-metric outcome. SHARP [2] governs reproduction of *existing* experiments; this paper governs autonomous *generation* of new ones.

Experiment tracking, AutoML, and NAS. Trackers such as MLflow and Weights & Biases [1, 18] record outcomes after a run exists but do not govern the agent loop or classify invalid edit attempts before training. AutoML and neural architecture search systems [6, 9] optimise within predefined search spaces; they are not designed to audit self-directed exploration, explain why a proposed change was kept or discarded, or expose lifecycle diversity. This paper makes the opposite tradeoff: a narrower set of controlled nodes with complete per-trial provenance.

Harness engineering and governance. The harness engineering literature independently validates this framing [3, 7, 11, 15, 17]: Trivedy frames the harness as everything around the model, including tools, sandboxes, memory, middleware, and execution hooks. Böckeler argues that trust in agent output requires constraining the solution space with computational guides and feedback sensors. This paper is not another task benchmark; it is a governance layer into which any benchmark task can be wrapped via a NodeSpec contract. The node specification is a computational guide; the pending-trial guard and state machine are computational sensors; memory injection is an inferential guide; the repeated-bad detector is an inferential sensor. Agent memory and scaffolding frameworks [5, 14] inform the design of context-resident memory injection and reflective self-improvement, but do not provide an evaluation protocol for governed autonomous experimentation.

3 Governed Control Plane

The system is a governed harness for autonomous ML experimentation. The manager proposes an experimental change; the worker materialises a patch and run artifacts. The control plane validates scope, executes the lifecycle, parses metrics, assigns a kept, discarded, or failed-invalid decision, updates memory, and appends the authoritative ledger record. Managers and workers are replaceable components; the control plane contract is fixed.

A worked trial. Before the full abstraction stack, we trace one governed trial end to end, drawn from the ResNet-trigger case study of Section 5.1. (i) *Proposal:* the manager emits one bounded change — summary, rationale, target parameter, intended effect — and the control plane assigns it a proposal provenance ID. (ii) *Scope validation:* the worker materialises a patch touching only `train.py`, the single whitelisted file; a patch touching frozen data or split logic would terminate here as `invalid_edit_scope`, before any training runs. (iii) *Execution under guard:* the control plane writes a pending-guard file, then the worker runs training; a crash here would leave a stale guard that forces a `failed_invalid` record at the next campaign start rather than a silent gap. (iv) *Metric parse and decision:* the parser extracts `val_auc` and the control plane — not the manager — compares it against the current best (this campaign improves 0.7747 → 0.7827 over 20 trials); an improvement is `kept`, a valid non-improvement `discarded`. (v) *Ledger append:* exactly one immutable JSONL record is appended — proposal text,

Table 1: Five auditability desiderata for governed agentic experimentation.

Desideratum	Operational meaning
D1. Bounded edit scope	The agent cannot change protected benchmark, data, or evaluation files.
D2. Complete attempt accounting	Every proposed attempt is recorded, including invalid proposals and crashes.
D3. Independent adjudication	Acceptance or rejection is decided by fixed control-plane rules, not by the agent.
D4. Provenance linkage	Each recorded attempt links to patch, configuration, run, metric, and decision evidence.
D5. Recomputability	Aggregate governance claims can be recomputed from the ledger without re-running training.

Table 2: Workshop evaluation concerns mapped to harness mechanisms.

Evaluation concern	Harness mechanism	Reported metric
Repeated poor behaviour	Append-only memory + repeated-bad detector	Repeated-bad rate
No ground truth for intermediate choices	Decision authority separated from manager; provenance IDs per trial	Provenance completeness
Deployment and scope safety	NodeSpec whitelist, no-op guard, pending-trial guard	Invalid-action rate
Long-horizon failure accumulation	State machine; terminal `kept`/`disc.`/`failed_inv.`	Failure taxonomy
Benchmark reproducibility	NodeSpec YAML, deterministic patch bridge, seeded replicates	Artifact manifest

patch reference and hash, raw-log reference, parsed metric, decision, failure category if any, and five provenance IDs — the guard is deleted, and the next budget slot may open. All 1,445 trials in this paper, including all failures, follow exactly this path; the only variation is which guard or comparison produces the terminal state.

To make auditability concrete, we operationalise it through the five desiderata in Table 1. These are not claimed to exhaust every possible failure mode of agentic experimentation, nor to be statistically independent axes. They define the minimum operational conditions needed for the claims in this paper to be inspectable. In particular, complete attempt accounting and provenance linkage are related but distinct: the former asks whether every attempted budget slot appears in the ledger, while the latter asks whether each recorded attempt carries enough evidence to inspect it.

3.1 Alignment with Agent Evaluation Criteria

Table 2 maps the Evaluation and Trustworthiness of Agentic AI workshop concerns to the harness mechanisms and metrics reported in this paper.

3.2 Manager, Worker, and Replacement Principle

The manager generates proposals (summary, rationale, target, intended effect). The worker executes bounded actions (patch + run artifacts). The control plane owns authority: validity, state transitions, budget, memory updates, provenance IDs, and final decision. This separation is the *replacement principle*: any manager backend — baseline heuristic, `prompt_manager`, `LangGraphManager`, or an AIDE-style tree-search — can be swapped without changing the governance contract. Two worker implementations span the reproducibility spectrum: `LocalWorker` applies "Change PARAM from

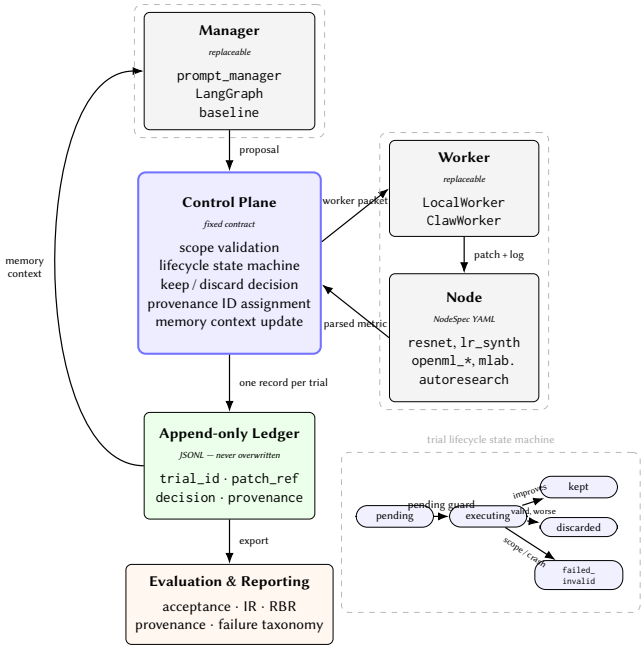


Figure 1: System architecture and trial lifecycle. *Replaceable components* (manager, worker, node) are interchangeable without changing the governance contract; the control plane contract is fixed across all six nodes and every manager backend tested in this paper. The control plane owns scope validation, lifecycle transitions, keep/discard decisions, provenance ID assignment, and memory context updates. The state machine inset (bottom right) shows that every opened budget slot produces exactly one terminal record—`kept`, `discarded`, or `failed_invalid`—appended to the append-only ledger before the next slot may open.

X to Y" directives with no external dependencies; `ClawWorker` invokes an external coding-agent path for free-form proposals, routing structured proposals through a deterministic constant-patch bridge that requires only Ollama.

3.3 Benchmark Node Inventory

Table 3 lists the six nodes reported in this paper. The control-plane contract is identical across all nodes: the only per-node configuration is the NodeSpec YAML (editable paths, frozen paths, run command, metric parser, and budget range).

3.4 Trial Lifecycle

Each trial follows the fixed lifecycle shown in Figure 1 (state machine inset). Every opened budget slot must produce a `kept`, `discarded`, or `failed_invalid` audit object.

The control plane writes a pending guard before worker execution, making mid-trial crashes auditable. A scope validator checks the patch against the NodeSpec whitelist before training begins: patches touching frozen files are rejected, recorded, and prevented from corrupting node state. The append-only JSONL ledger stores,

Table 3: The six benchmark nodes. All nodes share the same control-plane contract; only the NodeSpec YAML differs. Direction: max = maximise metric, min = minimise.

Node	Task type	Worker	Metric	Dir	Purpose in evaluation
`resnet_trigger`	Waveform classification	Claw	`val_auc`	max	Primary case study; memory ablation substrate
`lr_synthetic`	Logistic regression	Local	`val_score`	max	Cross-node transfer check; dependency-free
`openml_credit_g`	Tabular classification	Local	`val_auc`	max	Governance transfer to public tabular node
`openml_bank_mkt`	Tabular classification	Local	`val_auc`	max	Governance transfer with visible task improvement
`mlagentbench_vec.`	NumPy convolution	Local	`speed_score`	max	Benchmark compatibility probe
`autoresearch_linux`	LM pretraining	Claw	`val_bpb`	min	Real-task stress case; governance under high failure rate

Table 4: Trial lifecycle state machine. The pending guard is a file written before EXECUTING and deleted on any terminal transition; its presence at campaign start raises `PendingTrialError`, preventing concurrent writes to the append-only ledger.

From state	To state	Condition
PENDING	EXECUTING	Pending guard written; worker starts
EXECUTING	KEPT	Metric parses; beats current best; patch diff retained
EXECUTING	DISCARDED	Metric parses; does not beat current best
EXECUTING	FAILED_INVALID	Scope violation, runtime error, missing metric, no-op patch, or precondition failure
EXECUTING	FAILED_INVALID	Mid-trial crash detected via stale pending guard at next campaign start

Terminal states: KEPT, DISCARDED, FAILED_INVALID. All three produce a complete ledger record. Recovery tool: `scripts/recover_pending.py`.

per trial: proposal summary and rationale, patch and log references, parsed metrics, validity status, decision, failure category, provenance IDs, and reproducibility hashes.

Table 4 formalises the lifecycle. The key invariant is that every opened budget slot produces exactly one terminal record — there are no silent completions or hanging trials.

Lifecycle invariant. A trial begins in PENDING, then enters EXECUTING. It ends in exactly one terminal state: KEPT, DISCARDED, or FAILED_INVALID. The control plane maintains the following invariant:

> *For every budget slot that enters EXECUTING, the control plane appends exactly one terminal record—*`kept`*,* `discarded`*, or* `failed_invalid`*—to the append-only ledger before the next slot may open. No slot is silently skipped; no slot produces more than one record; no record is overwritten or deleted.*

Enforcement has two paths. *Normal path:* the pending guard is written before the worker starts and deleted on the terminal transition; any deviation raises an error before the ledger append. *Crash-recovery path:* if the process crashes mid-trial, the guard file persists; at the next campaign start `PendingTrialError` is raised, the stale guard is inspected via `scripts/recover_pending.py`, and a `failed_invalid` record is appended before any new trial begins. Neither path allows a budget slot to close without a ledger record.

The invariant is structural rather than empirical: it holds by construction of the control-plane code, not because nothing crashed during the reported experiments. The 100% provenance completeness observed across all 1,445 trials is its empirical certificate.

3.5 Failure Taxonomy and Metric Validation

Failure labels are assigned by the control plane at the terminal transition that writes the ledger record, not post hoc by the manager. The submitted evidence uses five observed `failed_invalid` categories: `invalid_edit_scope`, `no_op_patch`, `proposal_precondition_failed`, `runtime_error`, and node-level `invalid_config`. The schema also supports `metric_missing` and `edit_failed`; those labels do not appear in the primary 1,445-trial evidence set. Valid non-improvements receive `discarded`, not `failed_invalid`. The taxonomy is exhaustive for the current NodeSpec contract because every terminal transition is produced by one of those guards or by a parsed metric comparison. New nodes may add subcategories only if they map to the same three terminal outcomes before the ledger append. Structural validation uses forced stress campaigns, paired deterministic reruns, kept-artifact re-execution, and rule-based relabeling from observable ledger fields. Readers who want the taxonomy grounded before the aggregate results should consult Supplement Section E, which gives one concrete ledger example per category with its detection point and guard type.

3.6 Memory Modes and Decision Authority

The memory interface has three modes: no prior-trial context, append-only summary, and rationale-augmented summary. The rationale arm adds per-trial failure rationales and can truncate them with `rationale_max_tokens`. The ablation is a diagnostic probe: it tests whether the manager changes behavior when exposed to prior governed outcomes, and whether repeated-bad rate can detect that behavioral change. It is not treated as evidence that any memory format is generally superior.

The keep/discard decision lives in the control plane, not the manager. A valid candidate is kept only if the configured metric improves over the current best; a valid non-improving candidate is `discarded`; an invalid candidate is `failed_invalid` with a specific failure category. This deterministic, externally-owned decision rule follows practitioner guidance for long-running agents: harnesses should provide review loops and external checks rather than letting the model own success criteria [13].

4 Evaluation Protocol

The evaluation asks whether an autonomous ML experimentation loop is bounded, auditable, failure-aware, reproducible, and behaviorally affected by governance memory. Governance metrics are primary; task-metric AUC is secondary evidence that the loop runs end-to-end.

4.1 Benchmark Nodes

The primary scientific benchmark is `resnet_trigger`, a near-threshold detector waveform classification task with frozen H5 corpora, split logic, preparation code, and node metadata; workers may edit only

nodes/ResNet_trigger/train.py. The synthetic transfer benchmark, lr_synthetic, is a pure-NumPy logistic-regression classifier run via LocalWorker for dependency-free protocol validation; its data-generation constants are frozen.

We also instantiate two public OpenML tabular nodes: openml_credit_g (dataset 31) and openml_bank_marketing (dataset 1461). They expose only bounded config.yaml hyperparameters and model choices; dataset identity, target column, split logic, preprocessing, parser, and NodeSpec metadata are frozen. These are governance-transfer substrates, not competitive AutoML benchmarks.

The mlagentbench_vectorization node wraps a NumPy-convolution speed task as a bounded external-benchmark adapter. Benchmark code, correctness checking, and parser logic are frozen; only config.yaml edits over implementation choice and workload size are exposed. This is a compatibility probe, not a full MLAgentBench benchmark claim.

The sixth node, autoresearch_linux, is a GPT pretraining task (nanochat, Wikitext-103) on NVIDIA L40S hardware. Its metric is val_bpb (bits per byte, minimise), the paper's only minimise-direction task. Workers may edit only training configuration; architecture, dataset pipeline, and parser are frozen. Each trial takes approximately 6 minutes, so only none and summary arms are run: 4 independent seeds × 30 trials per arm = 240 trials.

4.2 Campaign Protocol

Controlled ablations are designed to start each campaign from a reset node state. The canonical scientific-node campaign uses prompt_manager with structured state-aware hyperparameter proposals and claw_style_worker with the deterministic patch path. The memory ablation runs equal budgets across three memory modes (none, summary, and rationale-augmented summary) under the same node, budget, parser, and reset discipline. The OpenML campaigns use LangGraphManager, summary memory, and 20-trial budgets on each public node. The MLAgentBench adapter uses LocalWorker and a 30-trial budget. Stress campaigns exercise forced failure paths at 100% rate. For every trial, the control plane records proposal, patch reference, raw log reference, parsed metrics, validity status, failure category, decision rationale, provenance IDs, and reproducibility hashes.

Implementation and hardware environment. All cross-node campaigns use deepseek/deepseek-v4-flash as the proposal manager via LangGraph, holding the model fixed to reduce the capability confound across nodes. Model-agnostic findings include provenance completeness, lifecycle correctness, and scope enforcement; proposal-quality findings may be manager-specific. ClawWorker campaigns ran on a server-class NVIDIA L40S host (CUDA 12.8, Ubuntu 22.04, PyTorch 2.x); LocalWorker campaigns ran on a CPU-only host under Python 3.11. The corrected ResNet L40S replication resets all editable node files to a verified frozen baseline before trial 1; older autoresearch_linux reset semantics are disclosed separately in Sections 5.6 and 6. Patch and run-log references are recorded with provenance IDs in the ledger; the public artifact contains the ledgers needed to recompute governance metrics, while raw execution-host artifacts are not required for the paper's primary claims. Full environment details and campaign IDs are provided in the supplement.

Table 5: Governance metric suite. Given N trials, these metrics characterise process reliability independent of task-metric outcome.

Metric	Formal definition	What it detects
Acceptance rate (AR)	$\lvert\text{kept}\rvert/N$	Valid improving edits
Invalid rate (IR)	$\lvert\text{failed\_invalid}\rvert/N$	Unevaluable trials
Repeated-bad rate (RBR)	Repeated-bad proposals $/N$	Redundant failure modes
Provenance completeness	Trials with all five IDs $/N$	Independent reproducibility
Artifact-reference coverage	Patch/log refs and hashes for materialized runs; classified records for pre-run failures	Ledger-level evidence traceability
Failure taxonomy	Counts per FailureCategory	Classified, not hidden, failures

4.3 Memory Ablation Design

As a diagnostic probe, the pre-stated behavioral hypothesis is:

$$\text{RBR(none)} > \text{RBR(summary)} > \text{RBR(rationale)}$$

where RBR is *repeated-bad rate*: the fraction of proposals repeating the same edit target and mechanism as a prior rejected trial without a new corrective rationale. Rationale-linked memory should reduce repeated-bad proposals more than raw summaries because it provides targeted failure signals rather than history noise [3, 11, 14, 16].

Two manager configurations are tested on ResNet-trigger: a deterministic prompt manager (design-constrained negative — round-robin makes memory structurally irrelevant) and a LangGraph manager calling qwen2.5-coder:7b at temperature 0.7 with an explicit AVOIDANCE RULE. All LangGraph proposals are parsed into structured parameter-change fields and applied via the deterministic patch bridge, eliminating coding-agent dependencies. Three ResNet-trigger replicates are run at budget 10; a four-arm verbosity ablation with 50-token rationales tests whether rationale length drives the result. The same three-arm LangGraph ablation is repeated on lr_synthetic as a cross-node transfer check.

Large-scale replication (NVIDIA L40S). To obtain variance estimates and test the pre-stated ordering, a third ablation configuration runs on an institutional GPU cluster: 3 memory modes × 5 independent seeds × 15 trials = 225 total trials. The manager is deepseek-v4-flash; the worker is qwen2.5-coder:7b at temperature 0.2. Each seed resets train.py before trial 1. Per-trial SHA-256 context hashes verify that memory accumulates across the 225-trial ledger.

4.4 Governance Metrics

Failure labels are control-plane-assigned. The categories observed in the submitted evidence include: runtime_error, invalid_edit_scope, no_op_patch, proposal_precondition_failed, and node-level invalid_config. The schema also supports metric_missing and edit_failed, although those labels are not part of the primary 1,445-trial evidence set. Valid-but-non-improving trials receive degraded_metric and are classified discarded, not failed

_invalid. A campaign with high invalid rate but 100% provenance completeness is more interpretable than one with unexplained score changes and no audit trail.

4.5 Metric Validation Probes

We run three lightweight validation probes for the governance metrics. First, two reruns of the MLAgentBench vectorization adapter test whether the same NodeSpec contract continues to produce complete lifecycle records under repeated runs. Second, a re-execution script rebuilds kept-trial state from cumulative patch references and reruns selected kept trials where the underlying execution artifacts are available. Third, we estimate structural relabeling agreement by computing Cohen's κ between control-plane-assigned labels and an independent rule-based relabeler applied to structurally observable fields only (scope-validation flags, patch presence, training-initiation signals, execution status). On a stratified sample of 50 failed_invalid trials spanning the observed validation categories, the structural relabeler achieves $\kappa = 0.830$ (95% CI [0.703, 0.958]). This is a structural consistency check, not a human inter-rater study. The remaining discordant records are edge cases where the ledger-level structural evidence does not distinguish the precise worker-side failure without consulting the referenced artifact file. Full human inter-rater validation remains future work.

5 Results

Results are presented in governance-first order. All numbers derive from the primary campaign ledgers in experiments/ledgers/; the submitted artifact also includes development and smoke ledgers that are excluded from the paper totals. All governance metrics (acceptance rate, invalid rate, repeated-bad rate, provenance completeness) are recomputable from the archived ledgers without rerunning any training; no GPU is required to verify the primary claims. The primary paper-relevant campaign set comprises 1,445 governed trials, indexed in the supplement. Across the primary set, the harness runs one real-worker scientific case study (resnet_trigger), one dependency-free synthetic transfer node (lr_synthetic), two public OpenML tabular nodes, one MLAgentBench adapter, and one real LM training node (autoresearch_linux); supplemental validation adds two one-trial stress campaigns. Together these campaigns exercise kept, discarded, failed-invalid, invalid-scope, no-op, invalid-configuration, and repeated-bad outcomes under a single ledger schema.

The section is staged as follows: the single-node case study (§5.1); the six-node governance view (§5.2); stress campaigns and the governed-vs-ungoverned counterfactual (§5.3); the memory-ablation diagnostic and its large-scale replication (§§5.4–5.5); and the real LM-training stress case (§5.6). Throughout, AR denotes acceptance rate, IR invalid rate, and RBR repeated-bad rate (Table 5).

5.1 Real-Worker Scientific Node Case Study

Table 6 summarises the 20-trial real-worker campaign on ResNet-trigger. Four valid improving edits are kept, five valid non-improvements are discarded with patch/log references and metric evidence retained in the ledger, and eleven impossible/no-op structured proposals are classified as proposal_precondition_failed before they can corrupt node state. Provenance is complete for all 20 records (100%).

Table 6: Real-worker scientific-node case study (kdd_resnet_scientific_20, ResNet-trigger, prompt_manager, claw_style_worker, budget=20).

Field	Value
Kept / disc. / failed-invalid	4 / 5 / 11
Non-kept categories	5 × degraded_metric; 11 × proposal_precondition_failed
Acceptance / invalid rate	0.20 / 0.55
Provenance completeness	1.00 (20/20)
Decision record completeness	1.00 (20/20)
Failure evidence completeness	1.00 (11/11 precondition failures classified)
Patch refs/hashes for materialized valid runs	1.00 (9/9 valid runs)
Initial / best val_auc	0.7747 / 0.7827 (+0.008022)
Baseline seed noise (5 seeds)	mean 0.785, bootstrap 95% CI of mean [0.775, 0.798]

The net AUC gain (+0.008022) remains within the baseline seed interval and should be treated as secondary evidence that the loop runs correctly, not as an optimisation claim.

5.2 Cross-Node Governance Transfer

Table 7 consolidates all six nodes into a single governance view. The central finding is that provenance completeness is 100% on every node without exception — including the 120-trial none-arm total failure and the s2 anomaly on autoresearch_linux, and across all three lifecycle outcomes (kept, discarded, failed-invalid).

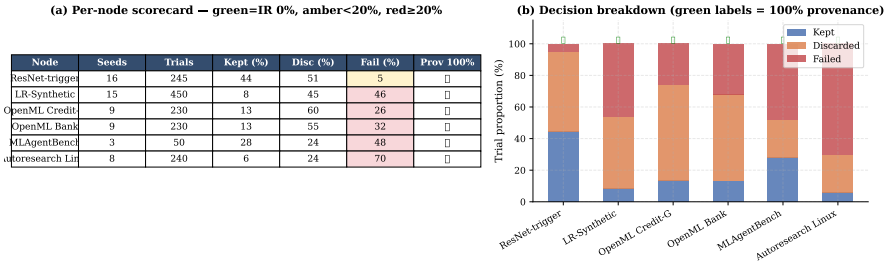
Public tabular nodes. openml_credit_g and openml_bank_marketing test whether governed keep/discard/fail decisions transfer to public tabular substrates (LangGraph manager, summary memory, b20 and b30 budgets). Both nodes expose only bounded config.yaml edits; dataset, split, preprocessing, and parser code are frozen. Provenance completeness is 1.00 across all 460 trials combined. openml_bank_marketing shows visible task-metric improvement (mean best AUC 0.9295 at b20, 0.9341 at b30; 95% CI $[0.9337, 0.9345]$ at b30, σ=0.0005), while openml_credit_g is a neutral transfer case (mean best AUC 0.7679 at b30). Invalid rates increase with budget as bounded candidates are exhausted or violate node-level configuration constraints; all such trials are classified as proposal_precondition_failed or invalid_config rather than silently dropped.

MLAgentBench adapter. The 30-trial main campaign records 2 kept, 4 discarded, and 24 failed-invalid trials with 1.00 provenance completeness. The high failed-invalid count reflects exhausted bounded candidates (proposal_precondition_failed), not benchmark failure; the control plane classifies them explicitly rather than inventing out-of-scope edits. Two additional 10-trial reruns add kept and discarded outcomes under the same NodeSpec contract, bringing the adapter evidence set to 50 trials with complete provenance.

5.3 Stress Campaigns and Ungoverned Counterfactual

Table 8 documents forced failure paths and the synthetic-node governance validation.

The scope-violation and no-op cases are backed by real ledger records. The lr_synthetic campaigns confirm that the governance



Node	Seeds	Trials	Kept (%)	Disc (%)	Fail (%)	Prov 100%
ResNet-trigger	16	245	44	51	5	▯
LR-Synthetic	15	450	8	45	46	▯
OpenML Credit-	9	230	13	60	26	▯
OpenML Bank	9	230	13	55	32	▯
MLAgentBench	3	50	28	24	48	▯
utoresearch Lin	8	240	6	24	70	▯

Figure 2: Governance overview across all six benchmark nodes. Top: per-node scorecard showing acceptance rate, invalid rate, and provenance completeness. Bottom: stacked decision breakdown (kept / discarded / failed-invalid) per node. Provenance completeness is 100% on every node regardless of task type, worker class, or failure rate—including the 120-trial total-failure arm on `autoresearch_linux`.

Table 7: Cross-node governance summary. All nodes achieve 100% provenance completeness regardless of task type, worker class, or failure rate. AR = acceptance rate, IR = invalid rate. ResNet-trigger appears as two rows because the campaign configurations differ fundamentally (prompt_manager vs. DeepSeek LangGraph; different memory modes). The autoresearch arms are listed separately because their AR/IR differ categorically. LR arm-level variation is summarised as a range across three memory modes.

Node	Trials	AR%	IR%	Lifecycle outcomes
`resnet_trigger` (case study)	20	20.0	55.0	kept, disc., failed_inv.
`resnet_trigger` (L40S, all arms)	225	46.7	0.4	kept, disc., failed_inv. (5 clean seeds verified)
`lr_synthetic`	450	8.4	46.2	kept, disc., failed_inv.
`openml_credit_g`	230	13.5	26.1	kept, disc., failed_inv.
`openml_bank_marketing`	230	13.0	32.2	kept, disc., failed_inv.
`mlagentbench_vec.`	50	28.0	48.0	kept, disc., failed_inv.
`autoresearch` (none arm)	120	0.0	100.0	failed_inv. only (total failure)
`autoresearch` (summary)	120	11.7	40.8	kept, disc., failed_inv.
Provenance completeness: 100% on every row above				

Table 8: Stress campaigns and synthetic-node validation. Each row is a silent failure in a naive unguarded loop that the harness makes visible.

Scenario	Campaign	Governed outcome	Evidence
Out-of-scope edit (frozen file touched)	`kdd_stress_scope`	Patch rejected before training; classified `invalid_edit_scope`; git state unchanged	1/1 trials; patch ref retained
No-op patch (byte-identical edit)	`kdd_stress_noop`	Training skipped; classified `no_op_patch`; empty patch hash recorded	1/1 trials; ledger record present
Synthetic baseline: `lr_synthetic` (logistic regression, LocalWorker)	`lr_synth_baseline`	All three lifecycle outcomes in 5 trials; provenance complete; NodeSpec YAML portable	2 kept, 1 discarded, 2 failed-invalid; best val_score = 0.923
Synthetic LangGraph transfer	`lr_synth_lg_*`	Same manager/memory ablation runs on a different node and worker type; all data-generation constants remain frozen	30/30 valid trials; RBR none/summary/rationale = 0.70/0.70/0.80

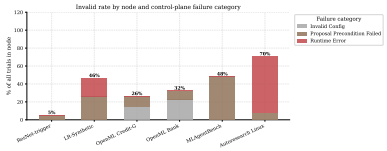


Figure 3: Failure taxonomy by node: share of `failed_invalid` outcomes per `FailureCategory`. Composition is node-specific: `autoresearch_linux` concentrates in `runtime_error`; tightly-constrained nodes accumulate `proposal_precondition_failed`.

protocol operates on a different ML task and worker type without code changes.

Ungoverned counterfactual. Every "the harness makes X visible" claim is only as strong as evidence that X would be invisible without governance. Across the 1,445-trial primary campaign set, governance classification acts on the majority of outcomes: 242 trials (16.7%) produce `runtime_error` records that the pending-trial guard and append-only ledger make auditable; 220 trials (15.2%) are caught by precondition checks before reaching a valid run; 85 trials (5.9%) expose node-level `invalid_config` failures; 662 valid-but-degrading trials (45.8%) are discarded by the control-plane decision rule. Without the pending-trial guard and append-only ledger, each `runtime_error` would be a silent filesystem event with no trial identifier, failure taxonomy, or provenance chain.

Table 9 reports matched governed/ungoverned pairs on two nodes chosen to span the outcome spectrum; the visual companion is Supplement Fig. S8. The ungoverned arm is an independent matched campaign, not a reconstruction from the governed ledger. It disables the pending-trial guard, failure-record appends, and blocking scope enforcement (Supplement B.1). A separate observation

log (`*_ungoverned_obs.jsonl`) records every executed trial and is the denominator for ledger completeness.

The two nodes bracket the failure spectrum. The `autoresearch_linux` row is an extremal, definitional case: when every trial fails and failure appends are disabled, completeness is necessarily zero. The informative mixed-outcome comparison is `openml_bank_mkt`, where the ungoverned ledger retains only the 7 valid trials and silently omits all 13 failures. Combined, 43 / 50 ungoverned trials (86%) leave no ledger record versus 0 / 50 governed; the 95% bootstrap CI for ungoverned completeness lies strictly below 1.00 on both nodes.

5.4 Memory Ablation as Diagnostic Probe

Memory context is evaluated as a diagnostic probe for governance behaviour, not as a proposed optimisation method. The pre-stated behavioral hypothesis is:

$$\text{RBR(none)} > \text{RBR(summary)} > \text{RBR(rationale)}$$

where RBR is *repeated-bad rate*: the fraction of proposals repeating the same edit target and mechanism as a prior rejected trial without a new corrective rationale.

Full arm-level results appear in Supplement Table S4. We use this ablation only as a diagnostic example for repeated-bad rate, not as a proposed memory method.

Primary diagnostic finding. On ResNet-trigger, the summary arm reduces repeated-bad rate in 2 of 3 replicates. A small cross-node `lr_synthetic` diagnostic (10 trials per memory mode) completes 30/30 valid LangGraph trials but does not reproduce the pre-stated RBR ordering: rationale (0.80) exceeds both none and summary (0.70), the reverse of the hypothesis. We interpret this reversal as an anchoring effect — richer rationale context gives the manager more prior failed patterns to fixate on rather than redirect from — confirming that repeated-bad rate is a useful per-node diagnostic rather than evidence of a general memory-mode effect.

Verbosity ablation. Truncating rationale to 50 tokens does not restore summary-level performance (RBR(C) = 0.778 = RBR(D) ≠ RBR(B) = 0.444). Rationale *length* is not the mechanism.

5.5 Large-Scale Memory Replication (NVIDIA L40S, 225 Trials)

The large-scale L40S replication (design and models in Section 4.3: 3 memory modes × 5 independent seeds × 15 trials = 225 trials) tests the full pre-stated ordering with variance estimates. Each seed reset `train.py` to a verified clean baseline before its first trial; reset integrity is confirmed by a shared `node_state_hash` at trial 1 across all 15 campaigns.

L40S replication: memory arms outperform none. Across all five clean seeds, mean best AUC follows none < summary ≈ rationale (0.899 vs. 0.939 and 0.941); the 0.002 difference between the two memory arms is not treated as meaningful. Cross-seed variability is lower for both memory arms, with $\sigma_{\text{none}} = 0.009$ vs. $\sigma_{\text{summary}} = \sigma_{\text{rationale}} = 0.002$ (4–5× reduction). Memory arms diverge from the no-memory arm by trial 5 and hold the gap through trial 15 (Supplement Figs. S4 and S9). An earlier run with an unverified reset showed node-state divergence at seeds s4–s5; all five seeds were rerun from a verified clean baseline. Provenance completeness: 100% across all 225 trials; the memory effect remains node- and manager-specific.

5.6 Real LM Training Node: Governance Under High Failure Rate

The `autoresearch_linux` node (GPT pretraining on Wikitext-103, NVIDIA L40S, metric `val_bpb`) is reported as a governance stress case under high failure rate. The none arm produces IR = 1.00 across all 120 trials (four seeds); three of four summary seeds produce valid improving trials (mean `val_bpb` reduction −0.062), while summary seed s2 also suffers total worker failure. Unlike the corrected ResNet replication, these campaigns were not run with the later frozen-baseline restore protocol, and the archived trial-1 hashes do not verify a common start state across all arms and seeds. We therefore report s2 as an unresolved infrastructure/protocol anomaly, not evidence for a memory effect. Across 240 trials including 151 runtime-error crashes, provenance completeness is 240/240 (100%): every `runtime_error` event is machine-readable with zero silent failures. The governance finding is classification fidelity under failure, not a general claim about memory-free managers.

6 Discussion and Limitations

Evidence strength. Table 10 separates demonstrated properties from case-study findings and non-claims.

Limitations. Four limitations bound the claims. *Scope of evidence:* six nodes broaden evidence beyond a single case study, but budgets are short and the RBR ordering holds in 2 of 3 ResNet replicates while reversing on `lr_synthetic`. *Label validation:* structural relabeling agreement (κ=0.830, Section 4.5) is a consistency check, not a human study; two-rater validation and holdout-node replication remain future work. *Task metric and reset scope:* the ResNet best AUC falls within the 5-seed baseline min–max range $[0.771, 0.811]$ (bootstrap 95% CI of mean: $[0.775, 0.798]$), so AUC gain is secondary evidence; the `autoresearch_linux` campaigns predate the frozen-baseline restore protocol, so their s2 anomaly is not used as memory-effect evidence. *Human utility:* the paper shows that governance properties can be enforced and measured, and what evidence is lost without them (Section 5.3) — not that governed ledgers improve human audit outcomes; a human-auditor utility study (do reviewers with governed ledgers detect seeded errors faster?) is the natural next step.

7 Conclusion

Across 1,445 governed trials on six ML nodes, the control plane classifies every outcome without silent corruption, with provenance guaranteed by construction and certified empirically. Autonomous ML experimentation should be evaluated on governance metrics—provenance, lifecycle classification, scope enforcement, and failure taxonomy—not only task metrics.

Table 9: Governed vs. ungoverned counterfactual comparison across two nodes. Ledger completeness = (trials with a ledger record) / (trials that ran). Governed completeness is 1.00 by structural invariant (Section 3.4); ungoverned completeness falls below 1.00 because `failed_invalid` trials are silently absent from the ledger. IR = invalid rate of the governed arm. 95% bootstrap CIs: 10,000 resamples, seed 42.

Node (profile)	N	Governed	Ungoverned	Drops	Interpretation
`autoresearch_linux` IR = 100%	30	1.00 [1.00, 1.00]	0.00 [0.00, 0.00]	30 (100%)	Total failure: ungoverned ledger is empty; campaign indistinguishable from one that never ran
`openml_bank_mkt` IR = 55%	20	1.00 [1.00, 1.00]	0.35 [0.15, 0.55]	13 (65%)	Mixed outcomes: ungoverned ledger retains only 7 valid trials; 13 classified failures leave no trace

Combined: 43 / 50 ungoverned trials (86%) silently absent; 0 / 50 governed trials absent (100% completeness by structural invariant).

Table 10: Evidence strength by claim.

Claim	Evidence level	Where
Control plane classifies kept / discarded / failed_invalid	Demonstrated	Secs 5.1–5.6
Append-only ledger: 100% provenance	Demonstrated	All nodes; Sec 5.2
Guard failures visible in audit record	Demonstrated	Stress, Sec 5.3
Protocol transfers across node types	Demonstrated, 6 nodes	Secs 5.2–5.6
Repeated-bad rate diagnoses memory sensitivity	Mixed case study	ResNet 2/3; `lr_synthetic` fails
Memory ordering none < summary ≈ rationale (AUC)	Observed, 5 clean seeds	Sec 5.5; not on `lr_synthetic`
Memory reduces cross-seed variance	Observed, 5 seeds	σ_{none}=0.009 vs. 0.002 (4–5×)
Rationale reduces repeated-bad rate	Not claimed	Non-monotonic; verbosity ablation not supported
No-memory arm: 100% invalid (this task/interface)	Demonstrated, 4 seeds	Sec 5.6
Governance classifies total worker failure	Demonstrated	Sec 5.6

References

[1] Lukas Biewald. 2020. *Experiment Tracking with Weights and Biases.* https://wandb.ai

[2] Joschka Birk, Gregor Kasieczka, Siddharth Mishra-Sharma, Benjamin Nachman, Dennis Noll, and Tanvi Wamorkar. 2026. *A Scientific Human-Agent Reproduction Pipeline.* arXiv:2604.18752 [hep-ph] doi:10.48550/arXiv.2604.18752

[3] Birgitta Böckeler. 2026. *Harness Engineering for Coding Agent Users.* Retrieved May 2026 from https://martinfowler.com/articles/harness-engineering.html

[4] Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Madry. 2024. *MLE-bench: Evaluating Machine Learning Agents on Machine Learning Engineering.* arXiv:2410.07095 [cs.CL] doi:10.48550/arXiv.2410.07095

[5] Pengfei Du. 2026. *Memory for Autonomous LLM Agents: Mechanisms, Evaluation, and Emerging Frontiers.* arXiv:2603.07670 [cs.AI] doi:10.48550/arXiv.2603.07670

[6] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural Architecture Search: A Survey. *Journal of Machine Learning Research* 20, 55 (2019), 1–21. https://www.jmlr.org/papers/v20/18-598.html

[7] Mitchell Hashimoto. 2026. *My AI Adoption Journey.* Retrieved May 2026 from https://mitchellh.com/writing/my-ai-adoption-journey

[8] Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. MLAgentBench: Evaluating Language Agents on Machine Learning Experimentation. In *Proceedings of the 41st International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 235).* PMLR, Vienna, Austria, 20271–20309. https://proceedings.mlr.press/v235/huang24y.html

[9] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren (Eds.). 2019. *Automated Machine Learning: Methods, Systems, Challenges.* Springer, Cham. doi:10.1007/978-3-030-05318-5

[10] Zhengyao Jiang, Dominik Schmidt, Dhruv Srikanth, Dixing Xu, Ian Kaplan, Deniss Jacenko, and Yuxiang Wu. 2025. *AIDE: AI-Driven Exploration in the Space of Code.* arXiv:2502.13138 [cs.AI] doi:10.48550/arXiv.2502.13138

[11] Ryan Lopopolo. 2026. *Harness Engineering: Leveraging Codex in an Agent-First World.* Retrieved May 2026 from https://openai.com/index/harness-engineering/

[12] Deepak Nathani, Lovish Madaan, Nicholas Roberts, Nikolay Bashlykov, Ajay Menon, Vincent Moens, Amar Budhiraja, Despoina Magka, Vladislav Vorotilov, Gaurav Chaurasia, Dieuwke Hupkes, Ricardo Silveira Cabral, Tatiana Shavrina, Jakob Foerster, Yoram Bachrach, William Yang Wang, and Roberta Raileanu. 2025. *MLGym: A New Framework and Benchmark for Advancing AI Research Agents.* arXiv:2502.14499 [cs.CL] doi:10.48550/arXiv.2502.14499

[13] Prithvi Rajasekaran. 2026. *Harness Design for Long-Running Application Development.* Retrieved May 2026 from https://www.anthropic.com/engineering/harness-design-long-running-apps

[14] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language Agents with Verbal Reinforcement Learning. In *Advances in Neural Information Processing Systems (NeurIPS '23, Vol. 36).* Curran Associates, Inc., New Orleans, LA, USA, 8634–8652. https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html

[15] Vivek Trivedy. 2026. *The Anatomy of an Agent Harness.* Retrieved May 2026 from https://www.langchain.com/blog/the-anatomy-of-an-agent-harness

[16] Chunlong Wu, Ye Luo, Zhibo Qu, and Min Wang. 2025. *Meta-Policy Reflexion: Reusable Reflective Memory and Rule Admissibility for Resource-Efficient LLM Agent.* arXiv:2509.03990 [cs.AI] doi:10.48550/arXiv.2509.03990

[17] Justin Young. 2025. *Effective Harnesses for Long-Running Agents.* Retrieved May 2026 from https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents

[18] Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, Fen Xie, and Corey Zumar. 2018. Accelerating the Machine Learning Lifecycle with MLflow. *IEEE Data Engineering Bulletin* 41, 4 (Dec. 2018), 39–45. https://people.eecs.berkeley.edu/~matei/papers/2018/ieee_mlflow.pdf

[19] Matei Zaharia, Omar Khattab, Lingjiao Chen, Jared Quincy Davis, Heather Miller, Chris Potts, James Zou, Michael Carbin, Jonathan Frankle, Naveen Rao, and Ali Ghodsi. 2024. The Shift from Models to Compound AI Systems. https://bair.berkeley.edu/blog/2024/02/18/compound-ai-systems/

[20] Qiran Zou, Hou Hei Lam, Wenhao Zhao, Yiming Tang, Tingting Chen, Samson Yu, Tianyi Zhang, Chang Liu, Xiangyang Ji, and Dianbo Liu. 2025. *FML-bench: Benchmarking Machine Learning Agents for Scientific Research.* arXiv:2510.10472 [cs.CL] doi:10.48550/arXiv.2510.10472

Supplementary Material: Evaluating Governed LLM-Driven ML Experimentation

Dowling Wong
dowling.wong@kit.edu
Karlsruhe Institute of Technology
Karlsruhe, Germany

Abstract

This supplement is a reproducibility appendix for the main paper. It provides ledger schema, NodeSpec contract examples, control-plane pseudocode, extended experiment tables, failure taxonomy with concrete ledger examples, campaign identifiers, hardware and model environment, and notebook details. No central claim in the main paper depends only on this supplement.

ACM Reference Format:
Dowling Wong. 2026. Supplementary Material: Evaluating Governed LLM-Driven ML Experimentation. In *Proceedings of KDD Workshop on Evaluation and Trustworthiness of Agentic AI (KDD ETAAI'26).* ACM, New York, NY, USA, 11 pages.

A Reproducibility and Artifact Description

A.1 Artifact and Code Availability

The code and ledger evidence bundle is submitted as supplemental material and is publicly available at https://github.com/dowlingwong/autoresearch_harness. The harness evidence snapshot used for the reported results is commit `ff3717b` (full SHA in `ARTIFACT_README.md`). The camera-ready paper source is available at https://github.com/dowlingwong/A-Governed-Harness-for-Auditable-LLM-Driven-ML-Experimentation; the submitted source archive and PDF are the authoritative camera-ready version. The harness is released under the MIT licence.

The artifact ZIP contains: `src/` (control-plane source), `configs/nodes/` (node YAML specs), `scripts/` (campaign runners), `tests/` (test suite), `experiments/ledgers/` (all paper trial records), `nodes/` (per-node training code), `paper_figures.ipynb` (figure notebook), `artifact_manifest.json` (campaign index), and `ARTIFACT_README.md` (step-by-step instructions). The supplemental bundle intentionally excludes large/private execution artifacts: the two ResNet HDF5 data files and execution-host patch/run-log files are not needed to recompute the reported governance metrics from the archived ledgers.

The submitted artifact contains the harness code, node configurations, submitted ledgers, and the analysis notebook. Execution-host patch and run-log artifacts are referenced by sanitized relative path fields and machine-readable provenance IDs in each ledger record; the raw execution-host paths are not included in the public artifact.

The top-level `artifact_manifest.json` indexes the submitted evidence bundle. Each trial record contains a trial identifier, campaign identifier, proposal summary/rationale, patch reference, raw-log reference, parsed metrics, terminal decision, failure category where applicable, node-state hash, and provenance IDs for proposal, patch, run, metric, and decision objects. Reviewers can map paper tables to ledgers under `experiments/ledgers/` and then to sanitized artifact references recorded in each ledger row.

A.2 Ledger Schema

Table S1: Core ledger fields. Terminal lifecycle states are `kept`, `discarded`, and `failed_invalid`; failure labels are attached only where a control-plane guard or worker failure produces a failure condition.

Field	Meaning
`trial_id`	Stable identifier for one budget slot.
`campaign_id`	Links rows to the campaign ID inventory in Section B.1.
`budget_index`	Trial index within the fixed campaign budget.
`decision`	Terminal state: `kept`, `discarded`, or `failed_invalid`.
`failure_category`	Control-plane label such as `runtime_error`, `invalid_edit_scope`, `no_op_patch`, `proposal_precondition_failed`, `invalid_config`, `metric_missing`, or `edit_failed`; valid-but-worse trials use `degraded_metric` and are `discarded`.
`proposal_summary`, `proposal_rationale`	Manager proposal text recorded before worker execution.
`patch_ref`, `patch_hash`	Patch artifact path and content hash when a patch is materialised.
`raw_log_ref`	Worker run-log artifact path.
`parsed_metrics`	Machine-parsed task metrics used by the decision rule.
`provenance`	Stable IDs for proposal, patch, run, metric, and decision objects.
`node_state_hash`	Hash of node-local state used for reset and divergence auditing.

A.3 Dry-Run Commands

The harness can be exercised without Ollama or a GPU using dry-run mode:

```
# Smoke test --- 3 trials, synthetic metrics, no LLM or node
      required
python3 scripts/run_campaign.py \
    --node resnet_trigger --campaign-id smoke \
    --budget 3 --dry-run

# LangGraph manager smoke test (stub LLM, no Ollama)
python3 scripts/run_campaign.py \
    --node resnet_trigger --campaign-id lg_smoke \
    --budget 2 --manager langgraph_manager \
    --dry-run --llm-stub
```

```
# Memory ablation dry run (all three modes)
python3 scripts/run_memory_ablation.py \
    --node resnet_trigger --budget 3 --dry-run
```

For a real campaign with a local Ollama model:

```
python3 scripts/run_campaign.py \
    --node resnet_trigger --campaign-id main \
    --budget 5 --manager prompt_manager \
    --memory-mode append_only_summary_with_rationale \
    --node-root nodes/ResNet_trigger \
    --model qwen2.5-coder:7b \
    --host http://localhost:11434
```

A.4 Hardware and Model Environment

Large-scale ResNet and `autoresearch_linux` campaigns ran on an institutional GPU cluster (NVIDIA L40S, CUDA 12.8, Ubuntu 22.04, PyTorch 2.x). Cross-node LangGraph campaigns use `deepseek-/deepseekv4flash` through the OpenAI-compatible DeepSeek API endpoint (`DEEPSEEK_BASE_URL`, default https://api.deepseek.com; authenticated via `DEEPSEEK_API_KEY`); campaigns were run in May 2026. The ResNet L40S replication uses `qwen2.5-coder:7b` at temperature 0.2. The stochastic memory ablation uses `qwen2.5-coder:7b` at temperature 0.7 with the explicit avoidance prompt and deterministic patch bridge. LocalWorker campaigns run under Python 3.11 on a CPU-only host.

Hosted API proposals may vary if the provider changes the served model behind a fixed identifier. Exact proposal text is most reproducible from the submitted ledgers. Governance metrics remain recomputable from archived trial records without API access. No API keys, account identifiers, or private endpoint secrets are included in the artifact.

A.5 Bootstrap Confidence Intervals

Seed-level bootstrap confidence intervals use 10,000 resamples with random seed 42. Resampling is performed over seed-level best metrics, not individual trial records, so each interval reflects between-seed variability.

A.6 Node-State-Hash Reset Verification

Each trial record contains a `node_state_hash` computed from the SHA-256 digest of the node's editable file set immediately before the worker executes. This field allows independent verification that every seed of a multi-seed ablation campaign started from an identical node state — a necessary condition for fair memory-mode comparisons. For the L40S replication (Table S5), all five seeds produce hash prefix `9c0118f33fcb` at `budget_index=1`:

```
python3 -c "
import json
for l in open('experiments/ledgers/<campaign_id>_trials.
     jsonl'):
    r = json.loads(l)
    if r['budget_index'] == 1:
        print(r['trial_id'], r['node_state_hash'][:12])
"
```

A divergence in the prefix across seeds indicates a reset failure and invalidates the ablation comparison for that seed. The hash is stored in every ledger record, so the check is replayable from the submitted artifact without re-running any training code.

A.7 Provenance and Artifact Completeness

Figure S1 shows per-node provenance completeness and artifact-reference capture rates by decision type. Panel (a) confirms 100% provenance completeness on all six nodes. Panel (b) shows that lower artifact-reference capture rates for `failed_invalid` trials are expected by design: trials that fail at the precondition stage (before worker execution) produce no patch diff, and trials that fail at runtime may produce no parsed metric. The provenance chain itself (proposal → patch → run → metric → decision IDs) is always complete regardless of the terminal outcome.

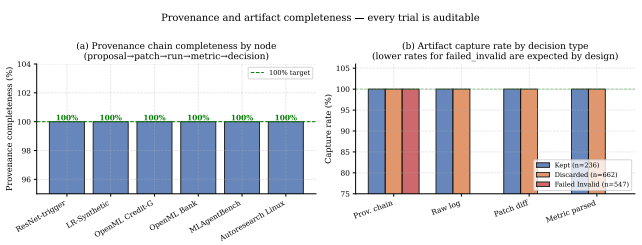


Figure S1: Provenance and artifact-reference completeness across all 1,445 governed trials. (a) Provenance chain completeness is 100% on every node. (b) Artifact-reference capture rates by decision type: lower patch and metric rates for `failed_invalid` are expected (pre-run failures produce no patch; runtime failures produce no parsed metric).

A.8 Analysis Notebook

The artifact bundle includes `paper_figures.ipynb`. The notebook loads submitted campaign ledgers from `experiments/ledgers/`, treats the append-only ledgers as the source of truth, and writes derived outputs to `paper/tables/` and `paper/figures/`. Install dependencies and run:

```
uv pip install -e ".[dev,notebook]"
# then: Kernel -> Restart & Run All
```

The notebook is an analysis convenience layer; the ledgers, not rendered plots, are the evidence source.

Selected notebook outputs (after final L40S data synchronisation): `fig1_governance_overview`, `fig3_ir_taxonomy`, `fig5_cross_node`, `fig6_provenance`, `fig7_openml`, and `fig9_resnet_linux_ablation`.

Retired or exploratory outputs (including the MLP synthetic diagnostic plot) should not be cited as submitted evidence unless the main paper explicitly restores that node to the evidence set.

B Extended Experiment Tables

B.1 Campaign ID Inventory

Table S2: Primary campaign identifiers for the 1,445-trial governed evidence set. One ledger file exists per listed seed. Campaign IDs with ranges (e.g. `s1–s5`) denote five separate JSONL ledgers.

Campaign ID(s)	Node	Arm / mode	Seed(s)	Budget
kdd_resnet_scientific_20	resnet_trigger	case study	–	20
deepseek_resnet_none_s1–s5	resnet_trigger	none	s1–s5	15
deepseek_resnet_append_only_summary_s1–s5	resnet_trigger	summary	s1–s5	15
deepseek_resnet_append_only_summary_with_rationale_s1–s5	resnet_trigger	rationale	s1–s5	15
deepseek_lr_none_b30_s1–s5	lr_synthetic	none	s1–s5	30
deepseek_lr_append_only_summary_b30_s1–s5	lr_synthetic	summary	s1–s5	30
deepseek_lr_append_only_summary_with_rationale_b30_s1–s5	lr_synthetic	rationale	s1–s5	30
deepseek_openml_cg_s1–s4	openml_credit_g	summary	s1–s4	20
deepseek_openml_cg_b30_s1–s5	openml_credit_g	summary	s1–s5	30
deepseek_openml_bm_s1–s4	openml_bank_marketing	summary	s1–s4	20
deepseek_openml_bm_b30_s1–s5	openml_bank_marketing	summary	s1–s5	30
mlagentbench_vectorization_main_30	mlagentbench_vec.	adapter main	–	30
deepseek_mlagentbench_s1, s2	mlagentbench_vec.	adapter rerun	s1–s2	10
deepseek_autoresearch_linux_none_s1–s4	autoresearch_linux	none	s1–s4	30
deepseek_autoresearch_linux_append_only_summary_s1–s4	autoresearch_linux	summary	s1–s4	30

Table S3: Validation and counterfactual campaign identifiers reported outside the 1,445-trial headline total. Ungoverned counterfactual arms append only valid-trial records to the ledger by design; the full trial denominators are recorded in the paired `*_ungoverned_obs.jsonl` observation logs (main-paper counterfactual table).

Campaign ID(s)	Node	Arm / mode	Seed(s)	Budget
kdd_stress_scope	resnet_trigger	scope stress	–	1
kdd_stress_noop	resnet_trigger	no-op stress	–	1
lr_synth_baseline	lr_synthetic	baseline manager	–	5
lr_synth_lg_none	lr_synthetic	none	–	10
lr_synth_lg_summary	lr_synthetic	summary	–	10
lr_synth_lg_rationale	lr_synthetic	rationale	–	10
kdd_cf_openml_gov	openml_bank_marketing	governed (matched)	–	20
kdd_cf_openml_ung	openml_bank_marketing	ungoverned (matched)	–	20
kdd_cf_arlinux_gov	autoresearch_linux	governed (matched)	–	30
kdd_cf_arlinux_ung	autoresearch_linux	ungoverned (matched)	–	30

B.2 LangGraph Memory Ablation Detail

Table S4: LangGraph memory ablation (qwen2.5-coder:7b, temp=0.7, avoidance prompt, deterministic patch bridge). RBR = repeated-bad rate. Arm C tests whether rationale *length* drives Arm D's result. The `lr_synthetic` rows use validation score, not AUC.

Arm	Memory mode	RBR	Kept	Disc.	Best metric	Observation
Primary ablation — rep1, budget=10						
A	none	0.78	1	9	0.7747	Baseline
B	summary	0.44	4	6	0.8212	Discovers `TRAIN_FRACTION`
D	full rationale	0.78	1	9	0.7747	Stuck on `BATCH_SIZE`
Verbosity ablation — same node/budget/model						
C	short rationale (50 tok)	0.78	1	9	0.7747	Matches D, not B: length is not the cause
Seed replication — rep2, budget=10						
A	none	0.78	1	9	0.7747	
B	summary	0.67	2	8	0.7903	Ordering fails in this replicate
D	full rationale	0.56	3	7	0.8220	Rationale arm best in rep2
Budget replication — rep3, budget=20						
A	none	0.84	2	18	0.8031	
B	summary	0.79	3	17	0.8150	Ordering holds; finds `TRAIN_FRACTION` at T5
D	full rationale	0.84	1	15+2	0.7747	Stuck on `TRACE_LEN`
Cross-node transfer — `lr_synthetic`, budget=10						
A	none	0.70	2	8	0.9230	Clean LocalWorker run
B	summary	0.70	2	8	0.9228	No RBR reduction on this node
D	full rationale	0.80	1	9	0.9227	Slightly higher RBR

B.3 Cross-Node Acceptance and Invalid Rates

Figure S2 summarises acceptance rate (AR) and invalid rate (IR) across all six nodes with 95% seed-level bootstrap confidence intervals. AR and IR vary substantially by node and manager interface. The governance claim is not that these rates are uniform, but that the lifecycle classification and provenance record are correct on every node regardless of the rates observed.

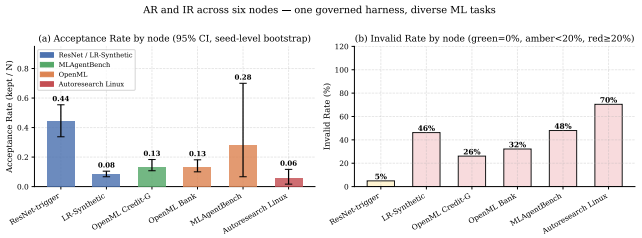


Figure S2: Cross-node acceptance rate (AR) and invalid rate (IR). Bootstrap CIs use 10,000 seed-level resamples. Colour denotes task family. AR and IR reflect proposal quality and task difficulty, not governance failures.

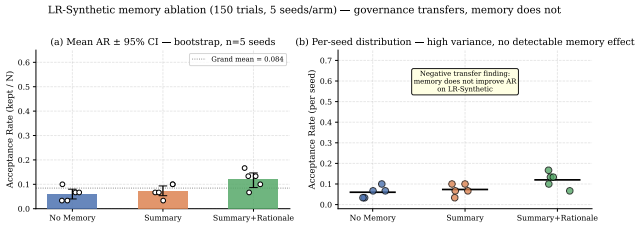


Figure S3: LR-synthetic memory ablation (150 trials, 5 seeds/arm). Mean AR and per-seed scatter show no detectable memory effect. Governance transfers; memory format does not.

B.4 LR-Synthetic Memory Ablation (Negative Transfer)

Figure S3 shows the memory ablation on `lr_synthetic`. No memory arm produces a detectable improvement in acceptance rate: the three arms are statistically indistinguishable across all five seeds. This is an explicit finding reported in the main paper (Section 5.4), not a framework failure — the control plane classifies every trial correctly in all three arms. Memory format is interface-specific; it does not transfer from the ResNet node to the synthetic logistic-regression node.

B.5 ResNet L40S Best-AUC Trajectory

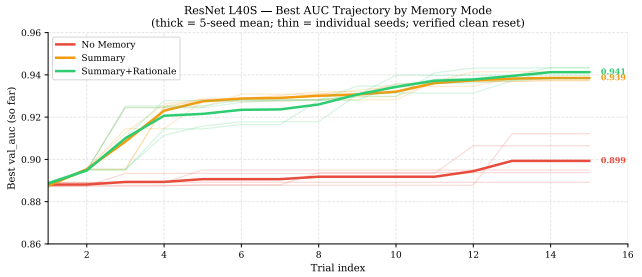


Figure S4: ResNet L40S best-AUC trajectory by memory mode (3 modes × 5 seeds, verified reset). Thick: 5-seed mean; thin: individual seeds. Memory arms diverge from the no-memory arm by trial 5 and hold the gap through trial 15, consistent with the 4–5× variance reduction reported in the main paper.

B.9 OpenML Dataset Statistics

The class imbalance in the bank-marketing dataset (88%) makes acceptance rate a more informative primary metric than raw accuracy; the control plane's keep rule (candidate > current best) is applied to `val_AUC`, which is robust to imbalance. Both datasets are publicly available on OpenML under the CC-BY licence.

B.6 Large-Scale L40S Replication

Table S5: Large-scale memory ablation (ResNet-trigger, NVIDIA L40S, 75 trials per arm = 3 modes × 5 seeds × 15 trials). Reset integrity confirmed by shared `node_state_hash` (`9c0118f33fcb`) at trial 1 across all 15 campaigns. σ is per-seed best-AUC standard deviation. 95% CIs use 10,000 seed-level bootstrap resamples (seed 42). Manager: `deepseek-v4-flash`; worker: `qwen2.5-coder:7b` at temperature 0.2.

Memory mode	Seeds	N	Acc%	Inv%	Mean best AUC	95% CI	σ
none	5	75	17.3	0.0	0.899	[0.892, 0.907]	0.009
summary	5	75	64.0	1.3	0.939	[0.937, 0.940]	0.002
rationale	5	75	58.7	0.0	0.941	[0.940, 0.943]	0.002

B.7 Autoresearch Linux Per-Seed Results

Table S6: Autoresearch Linux node (NVIDIA L40S, `deepseek-v4-flash` manager, `claw_style_worker`, 4 seeds × 30 trials per arm). AR = acceptance rate; IR = invalid rate; Gain = best − initial `val_bpb` (negative = improvement). Seed s2 of the summary arm suffered total worker failure; task-metric gain summaries use s1/s3/s4. AR uses the main-paper definition $|\text{kept}|/N$.

Arm	Seed	Kept	Disc	Fail	AR	IR	Gain
`none`	s1	0	0	30	0.00	1.00	—
	s2	0	0	30	0.00	1.00	—
	s3	0	0	30	0.00	1.00	—
	s4	0	0	30	0.00	1.00	—
`summary`	s1	4	17	9	0.13	0.30	−0.0616
	s2[†]	0	0	30	0.00	1.00	—
	s3	5	18	7	0.17	0.23	−0.0607
	s4	5	22	3	0.17	0.10	−0.0642

[†] s2 suffered total worker failure (27 `runtime_error` + 3 `proposal_precondition_failed`); no initial `val_bpb` was recorded. These campaigns predate the frozen-baseline restore protocol, and the archived trial-1 hashes do not verify a common start state across all arms and seeds; s2 is therefore an unresolved infrastructure/protocol anomaly, not memory-effect evidence. Governance correctly classified all 30 trials as `failed_invalid`.

B.8 OpenML Per-Budget Summary

Table S7: OpenML campaign summary by public node and budget. Seed-level bootstrap confidence intervals use 10,000 resamples (seed 42).

Node	Budget	Seeds	Trials	AR%	IR%	Mean best AUC	95% CI
`credit_g`	b20	4	80	20.0	8.8	0.7671	[0.7655, 0.7687]
	b30	5	150	10.0	35.3	0.7679	[0.7667, 0.7688]
`bank_mkt`	b20	4	80	20.0	17.5	0.9295	[0.9263, 0.9330]
	b30	5	150	9.3	40.0	0.9341	[0.9337, 0.9345]

Table S8: OpenML dataset characteristics for the two public classification nodes used in Section 5 of the main paper.

ID	Name	N	Feats.	Task	Maj. cls.
31	credit-g	1,000	20	Binary	70%
1461	bank-marketing	45,211	16	Binary	88%

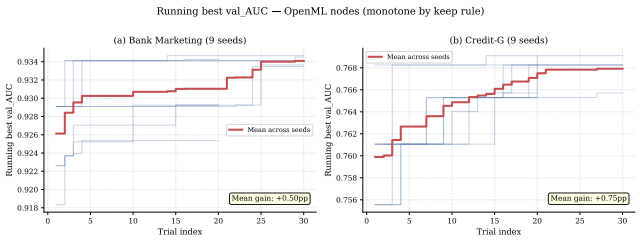


Figure S5: OpenML running-best val_AUC trajectories across all seeds. Monotone non-decrease is guaranteed by the control-plane keep rule. (a) Bank Marketing; (b) Credit-G.

B.10 OpenML Running-Best Trajectories

Figure S5 shows running-best `val_AUC` trajectories for both OpenML nodes. The step function is monotonically non-decreasing because the control plane accepts only valid improving trials — the keep rule is enforced structurally, not by the manager. Individual seeds (thin blue lines) and the 5-seed mean (red) show the variance compressed by the governed campaign discipline.

C Control-Plane Algorithm

C.1 Trial Lifecycle

Algorithm S6 gives the pseudocode for one governed trial. The control plane is the only writer to the append-only ledger; the manager never transitions state directly.

1. Write a pending-guard file for this budget slot.
2. Call **Manager.propose**(memory context, node spec) → proposal.
3. **Scope-check:** if proposal touches any frozen path → reject.
4. **No-op check:** if patch is empty or byte-identical → reject.
5. **Precondition check:** validate proposal fields → reject if malformed.
6. Call **Worker.execute**(proposal, node spec) → patch + raw log.
7. If worker fails → record `failed_invalid/runtime_error` or `edit_failed` and go to step 10.
8. Parse task metric from raw log.
9. If metric missing → `failed_invalid/metric_missing`. If metric ≤ current best → `discarded`. Else → `kept`; update current best.
10. Append one immutable ledger record (fields: proposal, patch ref, log ref, metric, decision, failure category, provenance IDs, node-state hash).
11. Delete pending-guard file.
12. Update append-only memory context from completed record.

Figure S6: Trial lifecycle pseudocode (one budget slot).

Steps 3–5 are pre-run guards: they prevent invalid edits from corrupting node state and produce classified failure records before any training occurs. Steps 7–9 are post-run classification: a valid but non-improving trial becomes `discarded`, not `failed_invalid`. The manager is not consulted at any classification step.

C.2 Control-Plane Trial Decision Timeline

Figure S7 illustrates the one-trial-one-record discipline on a representative campaign (OpenML Bank Marketing, seed 1). Every trial produces exactly one terminal record. The running best (step function) advances only when the control plane accepts a valid improving trial; it never decreases. This monotone property is structural: the keep rule `candidate > current_best` is evaluated and applied exclusively by the control plane.

C.5 Decision Rule

The decision function `decide_trial()` is purely functional with no side effects. It receives the candidate metric, the current-best metric, and a validity flag, and returns exactly one terminal state:

C.6 Memory Context Algorithm

The memory context is built by `build_memory_context()` and injected into the manager at each trial. Three modes produce qualitatively different inputs:

C.7 Repeated-Bad Rate Detection

The repeated-bad rate (RBR) is a governance metric that measures how often the manager repeats the same edit target and mechanism as a prior rejected trial without providing a new corrective rationale.

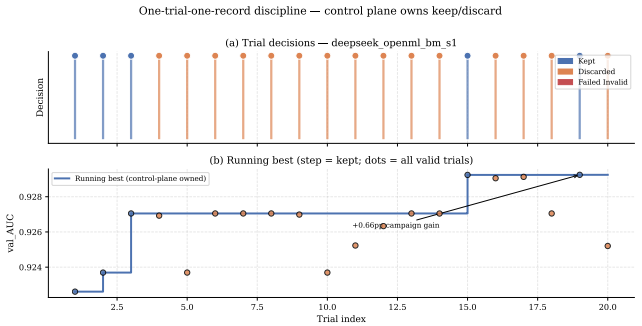


Figure S7: Trial decision timeline for one governed campaign (OpenML Bank Marketing, seed 1, 20 trials). Top: per-trial decision colour (kept / discarded / failed_invalid). Bottom: running best val_AUC step function and per-trial metric dots. The keep/discard decision is owned by the control plane, not the manager.

Definition. Let $P_i = (t_i, d_i)$ be the edit target and direction of proposal i (e.g., target = `LEARNING_RATE`, direction = *increase*). Proposal i is *repeated-bad* if:

(1) A prior proposal P_j, $j < i$, had the same target $t_j = t_i$ and direction $d_j = d_i$; and
(2) P_j received decision `discarded` or `failed_invalid`; and
(3) Proposal i does not include a rationale that references the prior rejection.

RBR = $|\{i : P_i \text{ is repeated-bad}\}|/N$ where N is the campaign budget.

Implementation. The detector in `memory/similarity.py` uses Jaccard similarity on unigram token sets of proposal summaries to identify near-duplicate proposals, then checks whether the candidate's summary token set S_i overlaps with any prior rejected proposal S_j above a threshold $\theta = 0.6$:

$$\text{Jaccard}(S_i, S_j) = \frac{|S_i \cap S_j|}{|S_i \cup S_j|} \geq \theta$$

Parameter direction is extracted by regex from the proposal rationale field (keywords: *increase*, *decrease*, *reduce*, *raise*, *lower*). Proposals that lack a detectable direction are not flagged.

C.8 Manager Prompt Template

The LangGraph manager (`manager/langgraph_manager.py`) builds a single-turn prompt from five components at each trial. Below is the template structure; bracketed tokens are filled at runtime.

```
You are a research manager proposing one bounded
hyperparameter change.

Node: [node_spec.name]
Description: [node_spec.description]
Metric to optimize: [metric_name] (direction: [direction])
Files you may instruct the worker to edit: [editable_paths]
Budget index (trial number): [budget_index]
Current best [metric_name]: [current_best_metric]

Editable values currently in [editable_file]:
  PARAM_A = current_value_A
  PARAM_B = current_value_B
  ...
```

C.3 Governed vs. Ungoverned Counterfactual Visual

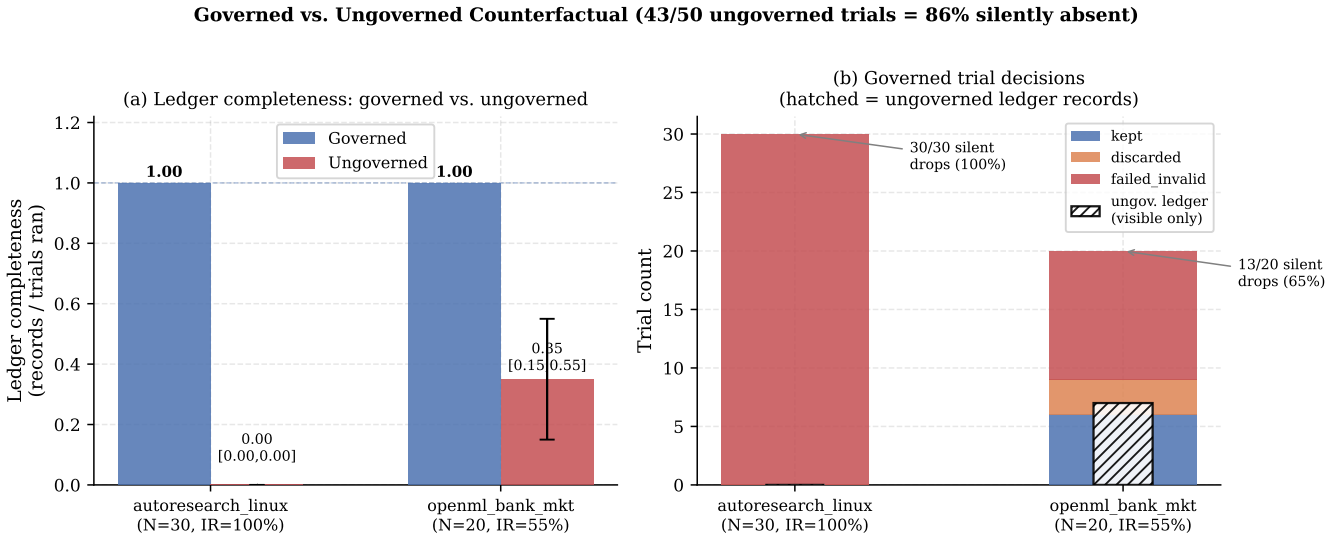


Figure S8: Governed vs. ungoverned counterfactual (visual companion to the main-paper counterfactual table). Left: ledger completeness with 95% bootstrap CIs; the governed CI collapses to $[1.00, 1.00]$ by structural invariant, while the ungoverned CI lies strictly below 1.00 on both nodes. Right: governed trial decision timeline with silent-drop positions annotated. Combined: 43 / 50 ungoverned trials (86%) leave no ledger record.

C.4 ResNet L40S Per-Seed Panels

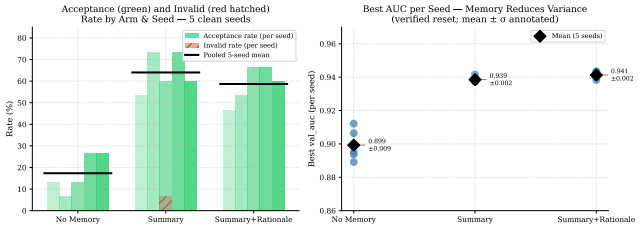


Figure S9: ResNet L40S memory ablation across five clean seeds with verified reset. Left: per-seed acceptance (green) and invalid (red) rates. Right: per-seed best AUC, showing $\sigma_{\text{none}} = 0.009$ vs. $\sigma_{\text{summary}} = \sigma_{\text{rationale}} = 0.002$ (4–5× reduction).

```
# --- if memory_context is non-empty: ---
Prior trial memory:
[memory_context.context_text]

AVOIDANCE RULE (mandatory): Study the trial history
above carefully. Identify every hyperparameter that
was already tried in the same direction (e.g.
increasing BATCH_SIZE, reducing LEARNING_RATE) and
produced no improvement or a failure. You MUST NOT
propose the same parameter change in the same
direction again. Choose a DIFFERENT hyperparameter
or a DIFFERENT direction. If every direction for a
parameter has been explored, switch to a different
parameter entirely.
```

if validity flag is FALSE (any pre-run or post-run guard fired) → return `failed_invalid` with the guard's `FailureCategory`.

elif `candidate_metric` > `current_best_metric` → return `kept`; update current best to `candidate_metric`.

else → return `discarded` with `failure_category = degraded_metric`.

The function never raises an exception and never reads or writes the ledger. All side effects (ledger append, memory update, best-metric update) happen in the control-plane campaign loop after `decide_trial()` returns.

Figure S10: Decision rule pseudocode (`control_plane/decision.py`).

```
# --- if memory_context is empty: ---
No prior trial memory available.

Propose exactly one bounded experiment. Respond with
ONLY a JSON object (no markdown, no extra text):
  "summary"  -- short slug e.g. "reduce-lr-2e-4"
  "rationale" -- one sentence on why this might help
  "objective" -- full worker instruction:
               "In FILE, change PARAM from X to Y.
                Edit only FILE."
  "param"    -- exact editable value name
  "old_value" -- current value as listed above
  "new_value" -- proposed replacement value

Constraints:
  - edit only: [editable_paths]
  - change exactly one listed value
  - param must be one of the editable values above
  - do not run the experiment; the harness runs it
```

Mode: none
Return a context containing only the NodeSpec parameters (editable symbols and their current values). No prior trial history is included. The manager has no information about past proposals or outcomes.

Mode: summary
Append one line per completed trial in chronological order:

```
Trial {i}: {decision} - {proposal_summary}
  metric: {parsed_metrics}
```

The manager receives the full history of proposals and outcomes, but with no explanation of *why* each trial succeeded or failed.

Mode: rationale (summary + rationale)
Same as summary, plus per-trial failure rationale lines:

```
Trial {i}: {decision} - {proposal_summary}
  metric: {parsed_metrics}
  rationale: {proposal_rationale}
  [AVOID: repeated {target} change - prior rejection]
```

The `[AVOID: ...]` warning is appended whenever the repeated-bad detector flags the current trial as semantically repeating a prior failed edit (see Section C.7). Rationale tokens can be truncated to a configurable `rationale_max_tokens` limit (50 tokens in the verbosity ablation).

Figure S11: Memory context algorithm (`memory/summarizer.py`).

The AVOIDANCE RULE is injected only when prior trial memory is non-empty (i.e., after the first trial). It operates on the memory context produced by the mode selected for that campaign arm. In the `none` mode the rule is absent; in the `summary` and `rationale` modes it is present and the manager is expected to honour it. The RBR metric (Section C.7) measures empirical compliance.

C.9 Scope Validation

The control plane's scope validator checks the file set touched by the patch against the NodeSpec's `editable_paths` whitelist and `frozen_paths` blocklist. A patch is rejected if it touches any file outside the whitelist or any file in the blocklist, regardless of the nature of the change. This fires before the worker executes training, so frozen files (dataset pipelines, split logic, evaluation harness) are never modified.

C.10 Crash-Recovery Path

If the process crashes mid-trial (e.g. OOM kill, host reboot), the pending-guard file persists on disk. At the start of the next campaign run, `PendingTrialError` is raised. The operator runs:

```
python3 scripts/recover_pending.py --list
python3 scripts/recover_pending.py --inspect \
    experiments/ledgers/<id>_pending.json
python3 scripts/recover_pending.py --mark-failed <id> \
    --reason "OOM during training"
```

This appends a `failed_invalid` record for the stale trial before any new trial begins, preserving the invariant that every opened budget slot produces exactly one terminal ledger record. The guard file is deleted only after the failure record is written.

D NodeSpec Contract Specification

D.1 Schema

A NodeSpec is a JSON-serialisable YAML file under `configs/nodes/`. The control plane reads it once at campaign start and treats it as immutable for the duration of the campaign. Table S9 lists the required and optional fields.

Table S9: NodeSpec YAML field schema. All fields are required unless marked optional.

Field	Type	Meaning
`name`	string	Unique node identifier used as campaign ledger key.
`description`	string	Human-readable node purpose.
`editable_paths`	list	Files the worker may edit; scope validator enforces this whitelist.
`frozen_paths`	list	Files and directories the worker must never touch; scope validator enforces this blocklist.
`run_command`	string	Shell command executed by the worker to train and produce output.
`metric_name`	string	Key expected in parsed metric output (e.g. `val_auc`).
`metric_direction`	string	`maximize` or `minimize`; determines the keep/discard rule.
`metric_parser`	string	Python import path to the parser function (`module:function`).
`acceptance_rule`	string	Expression evaluated by decision module (e.g. `candidate_metric > current_best_metric`).
`default_budget`	object	Default `trials` and `max_wall_clock_hours` (both optional at invocation time).
`setup_command`	string	Optional pre-campaign setup (e.g. `uv sync`).
`editable_symbols`	list	Optional: specific named symbols inside editable files (used by LocalWorker).
`failure_categories`	list	Exhaustive list of valid `FailureCategory` values for this node.

D.2 Annotated Example: `resnet_trigger`

This is the primary scientific node. The worker may only edit `train.py`; all other files including datasets, split logic, and architecture are frozen.

```
name: resnet_trigger
description: >
  Near-threshold detector waveform binary-classification
      benchmark
  with frozen H5 signal/noise traces and a single editable 1
      D ResNet
  training entrypoint.
editable_paths:
  - train.py
frozen_paths:
  - prepare.py
  - resnet_1d.py
  - signal_vacuum_sum_crop_4000x8000.h5
  - noise_traces_4000x8000.h5
  - artifacts/
  - data/
run_command: >
```

```
  ~/anaconda3/bin/python3 train.py > run.log 2>&1
metric_name: val_auc
metric_direction: maximize
metric_parser: >
  autoresearch.nodes.resnet_trigger.metric_parser:
      parse_val_auc
acceptance_rule: candidate_metric > current_best_metric
default_budget:
  trials: 50
  max_wall_clock_hours: 12
```

The metric parser converts `val_bpb` to `val_auc` via $1 - \texttt{val\_bpb}$ when both values are present, preferring the `val_bpb` reading. This indirection allows the same governance infrastructure to handle both maximise- and minimise-direction nodes.

D.3 Annotated Example: `lr_synthetic`

This is the dependency-free transfer node. The worker edits `train.py` but is further restricted to four named symbols; the data-generation constants are frozen by scope enforcement and by the `editable_symbols` hint used by LocalWorker.

```
name: lr_synthetic
description: >
  Synthetic binary-classification node (NumPy logistic
      regression).
  Dependency-free governance-protocol validation.
editable_paths:
  - train.py
editable_symbols:
  - LEARNING_RATE
  - REGULARIZATION
  - N_EPOCHS
  - BATCH_SIZE
frozen_paths:
  - artifacts/
run_command: python3 train.py
metric_name: val_score
metric_direction: maximize
metric_parser: >
  autoresearch.nodes.lr_synthetic
  .metric_parser:parse_val_score
acceptance_rule: candidate_metric > current_best_metric
default_budget:
  trials: 10
  max_wall_clock_hours: 1
```

Adding a new node requires only a YAML file in `configs/nodes/` and a registration entry in `src/autoresearch/nodes/registry.py`. The control-plane contract (lifecycle, guards, ledger format) is identical.

E Failure Taxonomy with Concrete Examples

The control plane assigns a `FailureCategory` label at the terminal transition that writes the ledger record. The taxonomy is exhaustive for the current NodeSpec contract: every `failed_invalid` outcome maps to exactly one of the categories below. Table S10 lists each category, its detection point, and a representative ledger excerpt.

E.1 Concrete Ledger Records

The following records are drawn verbatim from the submitted ledgers (provenance hash suffixes truncated for readability; full records are in `experiments/ledgers/`). Each record shows the seven fields that identify a trial's terminal outcome. The `metric_missing` category has not been observed in any real campaign to date; all zero-metric outcomes in the submitted artifact are classified as `runtime_error` (the worker exits non-zero before the metric parser runs).

Kept (campaign `kdd_resnet_scientific_20`, trial-001):


```
{"trial_id": "kdd_resnet_scientific_20-trial-001",
 "campaign_id": "kdd_resnet_scientific_20",
 "budget_index": 1,
 "decision": "kept",
 "failure_category": null,
 "proposal_summary": "lower-weight-decay",
 "parsed_metrics": {"val_auc": 0.774711}}
```


Discarded (campaign `kdd_resnet_scientific_20`, trial-002):


```
{"trial_id": "kdd_resnet_scientific_20-trial-002",
 "campaign_id": "kdd_resnet_scientific_20",
 "budget_index": 2,
 "decision": "discarded",
 "failure_category": "degraded_metric",
 "proposal_summary": "small-dropout",
 "parsed_metrics": {"val_auc": 0.773778}}
```


invalid_edit_scope (campaign `kdd_stress_scope`, trial-001):


```
{"trial_id": "kdd_stress_scope-trial-001",
 "campaign_id": "kdd_stress_scope",
 "budget_index": 1,
 "decision": "failed_invalid",
 "failure_category": "invalid_edit_scope",
 "proposal_summary": "lower-learning-rate",
 "parsed_metrics": null}
```


The scope guard fires before the worker executes; node state is never modified and training never runs.

no_op_patch (campaign `kdd_stress_noop`, trial-001):


```
{"trial_id": "kdd_stress_noop-trial-001",
 "campaign_id": "kdd_stress_noop",
 "budget_index": 1,
 "decision": "failed_invalid",
 "failure_category": "no_op_patch",
 "proposal_summary": "lower-weight-decay",
 "parsed_metrics": null}
```


proposal_precondition_failed (campaign `deepseek_autoresearch_linux_append_only_summary_s1`, trial-012):


```
{"trial_id":
   "deepseek_autoresearch_linux_append_only_summary_s1-trial-012",
 "campaign_id":
   "deepseek_autoresearch_linux_append_only_summary_s1",
 "budget_index": 12,
 "decision": "failed_invalid",
 "failure_category": "proposal_precondition_failed",
 "proposal_summary": "reduce-total-batch-size-32768",
 "parsed_metrics": null}
```


The precondition check fires when the manager produces a proposal whose field values fail validation (e.g. `new_value` is identical to `old_value`, or a required field is absent). No worker call is made.

runtime_error (campaign `deepseek_autoresearch_linux_none_s1`, trial-001):

Table S10: Failure taxonomy: detection point, guard type, and representative ledger evidence. All examples are drawn from real campaign ledgers in the submitted artifact; fields are formatted for readability.

Category	Fires at	Guard	Representative ledger evidence
`invalid_edit_scope`	Pre-run	Scope validator	Campaign `kdd_stress_scope`, trial-001: proposal attempts to edit `resnet_1d.py` (frozen); scope guard rejects; decision is `failed_invalid`; patch is never applied; node state is unchanged.
`no_op_patch`	Pre-run	No-op guard	Campaign `kdd_stress_noop`, trial-001: worker produces a patch that is byte-identical to the current `train.py`; no-op guard rejects; decision is `failed_invalid`; training is never executed.
`proposal_precondition_failed`	Pre-run	Precondition check	Campaign `deepseek_autoresearch_linux_append_only_summary_s2`, trials 28–30: manager produces malformed proposals after repeated `runtime_error` failures; precondition guard fires before worker call.
`runtime_error`	Post-run	Metric/exit check	Campaigns `deepseek_autoresearch_linux_none_s1–s4`, all 120 trials: coding agent fails to produce a valid patch on the LM pretraining task without memory context; worker exits non-zero; provenance IDs complete in all 120 records.
`invalid_config`	Post-run	Node validator	OpenML nodes reject hyperparameter values outside their bounded configuration domains, e.g. excessive tree depth or estimator counts. The run emits a machine-readable `invalid_config` label, which the control plane records as `failed_invalid`.
`metric_missing`	Post-run	Metric parser	Occurs when a run completes (exit zero) but the log file contains no parseable metric line, e.g. a training script that writes to `stderr` only or silently skips the validation epoch. Decision is `failed_invalid`; the raw log ref is retained for inspection.
`edit_failed`	Post-run	Worker path	ClawWorker only: the coding agent produces a patch file but fails to apply it cleanly (merge conflict, syntax error introduced by the patch). This is a worker-path artifact; it does not appear in LocalWorker or LangGraph deterministic-patch campaigns.
`degraded_metric`	Post-run	Decision rule	*Not* `failed_invalid`: valid run, metric parses, but does not beat current best. Decision is `discarded`. E.g. campaign `kdd_resnet_scientific_20`, trial-002: patch runs successfully, `val_auc` = 0.7738 vs. current best 0.7747; trial is discarded with full artifact retention.


```
{"trial_id":
   "deepseek_autoresearch_linux_none_s1-trial-001",
 "campaign_id": "deepseek_autoresearch_linux_none_s1",
 "budget_index": 1,
 "decision": "failed_invalid",
 "failure_category": "runtime_error",
 "proposal_summary": "reduce-embedding-lr-0.01",
 "parsed_metrics": null}
```


All 120 trials in the none-memory arm of the autoresearch-linux node share this pattern: the coding worker fails to produce a valid patch without memory context, exits non-zero, and the control plane records `runtime_error`.

invalid_config

(campaign `deepseek_openml_bm_s2`, trial-005):


```
{"trial_id":
   "deepseek_openml_bm_s2-trial-005",
 "campaign_id": "deepseek_openml_bm_s2",
 "budget_index": 5,
 "decision": "failed_invalid",
 "failure_category": "invalid_config",
 "proposal_summary": "increase-n-estimators-700",
 "parsed_metrics": null}
```


The `invalid_config` category covers node-level constraint violations where the edited file is in scope but the proposed value is outside the NodeSpec's bounded domain. It appears in the primary OpenML evidence set and is therefore included in the main-paper taxonomy.

E.2 Taxonomy Reliability Probe

To check that failure categories are structurally assignable from observable ledger evidence, we applied an independent rule-based relabeler to a stratified sample of 50 `failed_invalid` records spanning the observed validation categories. The relabeler uses only scope-validation flags, patch presence, training-initiation signals, and execution status — never the `failure_category` field itself. Cohen's κ between the control-plane assignments and the structural relabeler is $\kappa = 0.830$ (95 % CI $[0.703, 0.958]$), Landis–Koch "almost perfect" agreement. This is a structural agreement check, not a human inter-rater study. Perfect agreement ($\kappa = 1.0$) is achieved for `invalid_config`, `invalid_edit_scope`, and `proposal_precondition_failed`; the six discordant cases are `runtime_error` and `no_op_patch` edge cases where the distinguishing evidence (training seed, no-op message) is absent from the JSONL record but present in the referenced artifact files. Full human inter-rater validation remains future work.

E.3 Distinguishing Failed from Discarded

Key design choice: valid non-improving trials are `discarded`, not `failed_invalid`. This matters for audit:

- `failed_invalid`: a process problem (scope violation, coding failure, missing output) the operator should inspect.
- `discarded` trials indicate a valid experiment that did not improve the metric — a normal outcome in exploration campaigns with high discard rates.

Treating discard as failure would inflate the invalid rate and make it impossible to distinguish bad proposals from bad outcomes.

E.4 New Category Protocol

To add a new failure category:

(1) Add the enum label to `FailureCategory` in `memory/schemas.py`.
(2) Add detection logic in `control_plane/campaign.py` at the appropriate guard or post-run classification point.
(3) Add the name to `failure_categories` in the NodeSpec YAML(s).
(4) Add a test in `tests/test_trial_schema.py`.

The new category must map to `kept`, `discarded`, or `failed_invalid` before the ledger append.