Agent-Native Research Artifacts for AI Scientists

Anonymous Authors[1]

Abstract

AI Scientists already plan experiments, run code, and draft findings, yet they hand results back through a narrative paper, the same lossy medium humans use, which compresses a branching, iterative research process into a linear story and discards the majority of what was discovered along the way. This compilation imposes two structural costs: a **Storytelling Tax**, where failed experiments, rejected hypotheses, and the branching exploration process are discarded to fit a linear narrative; and an **Engineering Tax**, where the gap between reviewer-sufficient prose and agent-sufficient specification leaves critical implementation details unwritten. Tolerable for human readers, these costs become critical when AI agents must understand, reproduce, and extend published work. We introduce the **Agent-Native Research Artifact (ARA)**, a protocol that replaces the narrative paper with an agent-executable research package structured around four layers: scientific logic, executable code with full specifications, an exploration graph that preserves the failures compilation discards, and evidence grounding every claim in raw outputs. We complement the protocol with the **ARA ecosystem**, a coordinated set of agent skills—the *Live Research Manager* (LRM) that captures decisions and dead ends during ordinary development, the ARA *Compiler* that translates legacy PDFs and repos into ARAs, and the ARA *Seal*, a four-level review pipeline (analogous to a grammar checker for prose)—so artifacts are produced, imported, and verified automatically while human reviewers focus on significance, novelty, and taste. On PaperBench and RE-Bench, ARA raises question-answering accuracy from 72.4% to 93.7% and reproduction success from 57.4% to 64.4%. On RE-Bench's five open-ended extension tasks, preserved failure traces in ARA accelerate progress, but can also constrain a capable agent from stepping outside the prior-run box depending on the agent's capabilities.

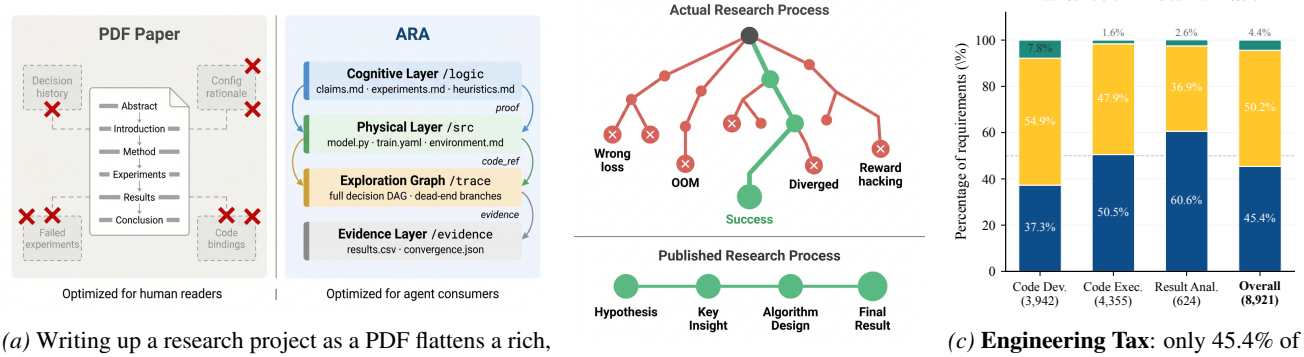
[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

1. Introduction

AI Scientists already plan experiments, run code, and draft findings, yet what they hand back is still a narrative paper that hides the work behind it. A research project is a branching object: months of hypotheses tested and rejected, implementation tricks found by trial and error, design alternatives weighed, and the full trajectory that explains why the final approach won. Writing it up as a paper (Medawar, 1963; Canini, 2026) flattens that object into a linear story and drops the failed experiments, the tacit engineering knowledge, and the branching process along the way (Rosenthal, 1979; Franco et al., 2014). The loss was tolerable when every reader of a paper was human; it is not when AI agents now read papers to understand a field, reproduce experiments to validate findings, and extend prior methods to new settings (Lu et al., 2024; Liu, 2026a), since each of those tasks needs precisely what the write-up discards (Figure 1a). The loss decomposes into two structural costs.

The first is the **Storytelling Tax**: the systematic erasure of research process knowledge that happens when work is written up as a story (Figure 1b). Research does not proceed linearly; it branches, backtracks, and accumulates hard-won failure knowledge before converging on a publishable result (Kuhn, 1962; Medawar, 1963). The paper flattens that process into a polished linear story and drops every failed experiment, rejected hypothesis, and abandoned approach in the process. The bias toward success leaves failures undocumented: modern platforms archive the final artifact, but the branching process remains unrecorded, so groups independently rediscover the same dead ends (Rosenthal, 1979; Franco et al., 2014). Our analysis of the METR eval-analysis-public dataset (Wijk et al., 2025), covering 24,008 agent runs across 21 frontier models on RE-Bench, quantifies the cost (per-task breakdown in App. A.3): failed runs account for **90.2%** of total dollar cost (and 59.2% of tokens), with a median failed-to-success token ratio of **113**×; without those failure records, every agent must rediscover every dead end. Equally lost is the record of human *judgment*: every rejection, revision, and endorsement along the way



(a) Writing up a research project as a PDF flattens a rich, branching object into a lossy narrative; ARA preserves it as an agent-executable package.

(b) **Storytelling Tax**: branching exploration collapses into a linear story.

(c) **Engineering Tax**: only 45.4% of 8,921 PaperBench requirements are PDF-sufficient.

Figure 1. The two structural costs of PDF-based research, and ARA's response.

is a preference signal over what counts as good research, the scarce resource that binds once agents take on the grunt work. The write-up discards that signal; a preserved trajectory turns it into structured supervision that compounds across projects.

The second is the **Engineering Tax**: the gap between *reviewer-sufficient* and *agent-sufficient* documentation (Figure 1c). The paper communicates its contribution at the level of detail needed to convince a human reviewer; the codebase provides an implementation but not the operational specification needed to execute it. Between the two lies tacit knowledge (Polanyi, 1966) (algorithmic tricks, implementation decisions, configuration choices), which exists in no written document and is transmitted only through direct lab contact or painstaking reverse-engineering. We quantify this void by classifying each of PaperBench's 8,921 expert-annotated reproduction requirements across 23 ICML 2024 papers (Starace et al., 2025) against its source PDF (per-category breakdown and gap-type taxonomy in Appendix A.2): despite widespread artifact sharing, only **45.4%** are fully specified. Code development is the most underspecified category (37.3% sufficient), and missing hyperparameters alone account for 26.2% of all gaps (full breakdown in Appendix A.2): a fundamental mismatch between the precision at which papers are written (sufficient to produce belief) and the precision at which agents must operate (sufficient to produce correct execution) (Stodden et al., 2016; Baker, 2016).

Both taxes persist because the human reader has always been the bandwidth-limited layer; capable AI agents now read at machine bandwidth on the reader's behalf, and three trends point at what an agent-native artifact should look like. AI agents co-author code, run experiments, and iterate on hypotheses alongside researchers (Lu et al., 2024), with LLM adoption associated with paper-production increases of 23.7%–89.3% across scientific fields (Kusumegi et al., 2025), so the full trajectory already exists as machine-readable text yet no protocol preserves it. Humans skim (Renear and Palmer, 2009) while agents benefit from exhaustive detail, so one artifact cannot serve both. And research is scaling into a parallel enterprise where agents fork, extend, and merge each other's work; narrative PDFs cannot, but a structured artifact can, letting research compound like software.

Existing efforts address fragments: FAIR data (Wilkinson et al., 2016), RO-Crate (Soiland-Reyes et al., 2022) archival bundles, Nanopublications (Groth et al., 2010) atomic claims, and `AGENTS.md` (OpenAI, 2025) agent-oriented code documentation. None jointly structure scientific logic, executable code, and exploration history into an operable object (§K).

We propose the **Agent-Native Research Artifact (ARA)**, a protocol that recasts the primary research object from narrative document to agent-executable knowledge package, with papers as compiled views (Figure 1a). ARA organizes research into four interlocking layers: *structured scientific logic* (queryable claims and dependency graphs), *executable code* with full operational specifications, an *exploration graph* preserving the branching research process (failed experiments, rejected hypotheses, design pivots), and *grounded evidence* binding every claim to its raw outputs. The artifact is designed not to be read, but to be *operated*: an agent queries structured claims, executes declarative specifications, and builds on the full decision history directly.

We complement the protocol with an **ecosystem** (§3) of three coordinated agent skills: the *Live Research Manager* (LRM) silently captures decisions and dead ends during AI–human research, the ARA *Compiler* converts legacy PDFs and repositories into protocol-conforming ARAs, and the ARA *Seal* review pipeline automates structural verification, argumentative-rigor auditing, and budget-aware reproduction (a grammar checker for prose), redirecting expert atten-

tion from mechanical checking to judgment, novelty, and taste (Aczel et al., 2021).

2. The ARA Protocol

The ARA protocol defines a file-system ontology that materializes the four kinds of knowledge a research project produces as four queryable layers (Figures 2, 3).

2.1. Design Principle

ARA rests on one principle, **Knowledge over Narrative**: the evolving knowledge produced during research is the primary scientific object; the paper is a compiled view. An agent engaging with that knowledge asks four structurally distinct questions, *why* it works, *how* it is implemented, *what was tried*, and *what the numbers are*, whose answer types conflict (stable reasoning vs. iterating code; branching trajectory vs. linear prose; machine-precise evidence vs. rounded summaries). ARA materializes each as its own layer (Figure 3): plain-text, independently queryable directories that agents navigate with standard tool calls, with *progressive disclosure* so finite context windows are not flooded.

2.2. ARA Architecture

A manifest `PAPER.md` provides a YAML frontmatter and layer index that lets an agent triage relevance in $\sim$500 tokens (App. A.5); the four layers below decompose the artifact along the ten reproduction-critical information categories distilled from PaperBench rubrics (App. A.1).

Cognitive (`/logic`). The *why*: `problem.md` states the gap and insight; `solution/` specifies architecture, algorithm, and convergence-critical heuristics; `claims.md` lists falsifiable assertions with proof pointers; `experiments.md` declares the verification plan; `related_work.md` replaces passive citations with *typed dependencies* (`imports` inject prior definitions, `bounds` propagate constraints, `baseline` enables regression detection).

Physical (`/src`). The *how*, calibrated to contribution type. Algorithmic work uses *kernel mode* (core modules with typed I/O, often 10–100$\times$ smaller than the full repo, with boilerplate regenerated by the agent on demand); systems work uses *repository mode* with an `index.md` mapping each file to the ARA component it implements. `configs/` annotates every hyperparameter with rationale and search range; `environment.md` pins deps, hardware, and seeds (App. A.4).

Exploration Graph (`/trace`). `exploration_tree.yaml` stores the research DAG along the seven typed events emitted by the LRM (§3.1): `decision`, `experiment`, `claim`, `heuristic`, `dead_end`, `pivot`, `observation`. Nesting encodes parent→child edges; `also_depends_on` captures convergence; `claim` and `heuristic` link to canonical entries in `/logic`. The format is a "git log for research": dead-end nodes preserve the hypothesis, failure mode, and lesson that narrative papers discard (Fig. 2).

Evidence (`/evidence`). Outputs only: `results/` holds machine-readable metric tables with exact values and source annotations; `logs/` holds curves, resource usage, and diagnostics. Every claim's proof chain flows `claims.md` → `experiments.md` → `/evidence/`. Because experiment *logic* lives in `/logic` and experiment *data* lives exclusively in `/evidence`, a verifier granted only the kernel and logic cannot fabricate by copying expected values, and the same separation makes every ARA a drop-in training environment (task in `/logic/experiments.md`, reward in `/evidence/`, preference signals in `/trace/`).

We say an ARA is *sufficient* when it entails its core claim under standard execution: every input, specification, and decision required to derive the claim is contained in the artifact. Sufficiency is a property of the artifact, not of any particular agent.

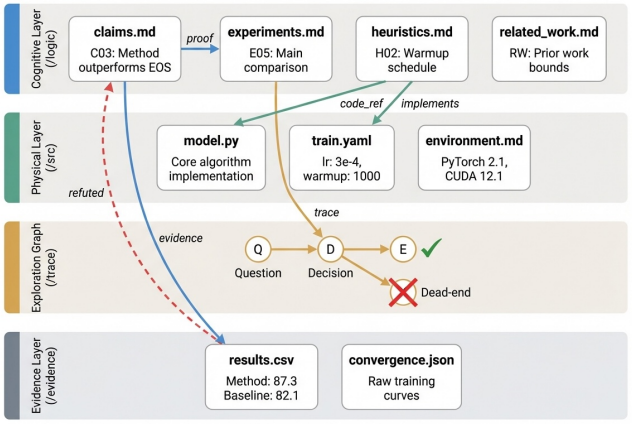


Figure 2. Cross-layer structure of ARA.

```
my-research-ara/
|-- PAPER.md            -- Manifest + layer index
|-- logic/    [COGNITIVE LAYER]
|   |-- problem.md      -- Gap, insight, assumptions
|   |-- claims.md       -- Falsifiable assertions
|   |-- experiments.md  -- Verification plans
|   |-- solution/       -- architecture, algorithm
|   `-- related_work.md -- Typed dependency graph
|-- src/      [PHYSICAL LAYER]
|   |-- configs/        -- Hyperparameters + bounds
|   |-- kernel/ (repo/) -- Core code or full repo
|   `-- environment.md  -- Versions, deps, hardware
|-- trace/    [EXPLORATION GRAPH]
|   `-- exploration_tree.yaml -- DAG with dead ends
`-- evidence/ [EVIDENCE LAYER]
    |-- results/        -- Metrics, scores, outputs
    `-- logs/           -- Curves, traces, resources
```

Figure 3. The ARA directory structure.

3. The ARA Ecosystem

The ARA schema is too rich to populate by hand; agent-native generation is feasible only because research agents absorb the authoring, verification, and rendering work as a byproduct of normal research (Figure 4). Three coordinated agent skills constitute the ecosystem: the **Live Research Manager** (LRM, §3.1) silently crystallizes new ARAs from researcher–agent sessions; the **ARA Compiler** (§3.2) converts legacy PDFs and repositories into protocol-conforming artifacts; and the **ARA Seal** (§3.3) translates the artifact's machine-checkable structure into a verifiable credential gating downstream use. Full specifications: App. C.1, B.2, D.3.

3.1. Live Research Manager

AI-native research is *born digital and born textual*: every instruction, intermediate result, design choice, and abandoned direction already exists as machine-readable text in the researcher–agent conversation. Prior efforts to preserve process knowledge (negative-result journals (Matosin et al., 2014), registered reports (Chambers, 2013)) foundered because documentation remained a separate, unrewarded burden; AI-native research dissolves that barrier.

The **LRM** is an *agent skill* (Anthropic, 2025a): a natural-language specification loadable by any coding agent, no custom SDK.[1] It stays silent during research, then runs a retrospective three-stage pipeline at each session boundary: a *Context Harvester* scans the session record for research-significant events; an *Event Router* classifies each into seven typed categories (decision, experiment, claim, heuristic, dead_end, pivot, observation), tags provenance (`user`, `ai-suggested`, `ai-executed`, `user-revised`), and writes it to the right ARA layer; a *Maturity Tracker* promotes staged observations only when closure signals (abandonment, affirmation, empirical resolution, commit) settle them. The LRM is stateless; the artifact carries memory across sessions (App. C).

[1] ARA components (LRM, Compiler, Seal with Rigor Auditor) and a hosted reference instance will be open-sourced upon camera-ready; URL withheld for double-blind review.

3.2. ARA Compiler: Recovering the Legacy Record

The born-agent pathway produces the highest-fidelity ARAs but cannot reach backward. The **ARA Compiler** is a many-to-one agent skill that translates legacy PDFs, code repositories, expert rubrics (e.g., PaperBench (Starace et al., 2025)), and trajectory logs (e.g., RE-Bench MALT traces (Wijk et al., 2025)) into a protocol-conforming ARA with graceful degradation: a PDF alone yields stub layers; richer inputs populate more.

The core problem is not extraction but *forensic reconstruction* of the cross-layer bindings narrative compresses away (claim→experiment→evidence→code). The Compiler proceeds top-down through Semantic Deconstruction (strip narrative), Cognitive Mapping (populate `/logic`), Physical Grounding (generate `/src`; when a repo is available, code-paper reconciliation surfaces tacit knowledge (Li et al., 2026) as provenance-tagged heuristics), and Exploration Graph Extraction (DAG with dead-end leaves). Quality is enforced by an in-loop ARA Seal Level 1 validator (§3.3); across all 30 compilation runs in our corpus, generate → validate → fix converges within 3 rounds (App. J.1). When a library of compiled ARAs exists, *collective inference* retrieves same-domain heuristics and adds them as `collective_inference`-tagged candidates (App. B).

3.3. ARA Seal: Verification and Review

LRM and the Compiler keep producing new ARAs; before any agent or human invests compute on one, they need a cheap way to know which artifacts to trust. Today that trust

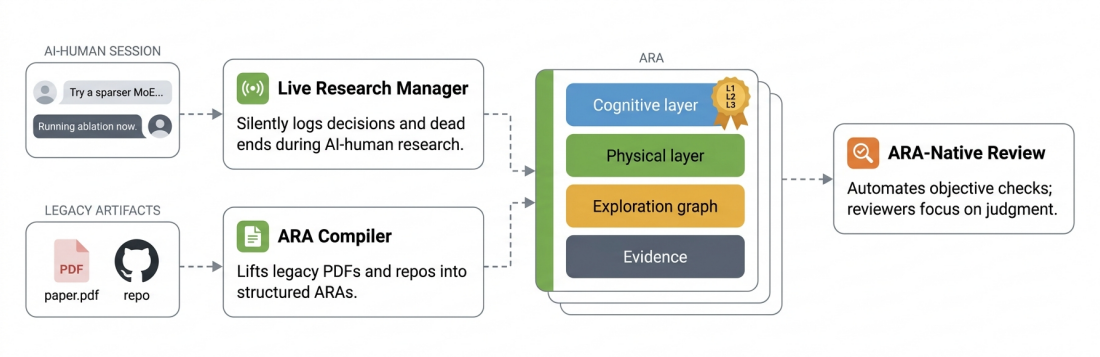


Figure 4. The ARA ecosystem: how research is produced and reviewed against one canonical artifact.

is bought with the scarcest resource in scientific evaluation, expert human attention: review loads have outgrown the reviewer pool (Aczel et al., 2021), and reviewer bandwidth is increasingly consumed by mechanical verification ("does the code run?", "does Table 3 support Claim 2?") rather than the substantive judgment only domain experts can provide. Because an ARA *is* structured, machine-executable data, those mechanical checks become objectively verifiable, and higher-order properties become *machine-assessable*: automated checks provide probabilistic evidence that informs human reviewers rather than replacing them.

We define the **ARA Seal** as a machine-verifiable credential with four escalating levels, each gating the next (per-level implementation and rubric anchors in Appendix D.3; effectiveness evaluation in Appendix J). **Level 1 (Structural Integrity)** is deterministic and runs in seconds: the directory ontology exists, every structured file conforms to its schema (each claim carries `Statement`, `Status`, `Falsification criteria`, `Proof`; each heuristic carries `Rationale`, `Sensitivity`, `Bounds`), and all cross-layer references resolve. **Level 2 (Argumentative Rigor)** invokes a *Rigor Auditor* agent that, without executing code, scores the artifact along six anchored dimensions (evidence relevance, falsifiability quality, methodological rigor, scope calibration, argument coherence, exploration integrity), each on a 1–5 rubric, with findings tagged by severity; we stress-test the auditor with a 115-injection mutation benchmark in App. J.2. **Level 3 (Execution Reproducibility)** runs scaled-down directional checks (small data, few epochs) on claims selected by criticality, with the verifier isolated from the evidence layer to prevent fabrication; full-scale reproduction is post-acceptance or community-driven. **Level 4 (Human Judgment)** is the only level that cannot be automated: reviewers receive the submission alongside the CI Report (Levels 1–2) and Empirical Review Report (Level 3), and their role shifts from verification to judgment on significance, novelty, taste, and problem framing. Passing the applicable levels issues a **Seal Certificate**: a signed record of artifact ID, level, timestamp, environment hash, and per-claim outcomes, that downstream agents check before investing compute. Full Seal evaluation, mutation benchmarks, and implementation details are in Appendix J.

Together, these three skills close a loop in which the ARA is the only persistent state shared between humans and research agents. The producer and consumer dynamics of this human–AI interface, and how contributions compound across artifact forks, are detailed in Appendix E.

4. Evaluation

We evaluate ARA across three layers of increasing research ambition: *understanding* (§4.1, can an agent extract knowledge from the artifact?), *reproduction* (§4.2, can it execute research from the artifact?), and *extension* (§4.3, can it build on prior work more efficiently with failure knowledge?). All three layers share a single comparison: holding the agent, the task, and the ground truth fixed, does an ARA compiled from a research project's source materials outperform the conventional PDF+repo? Two benchmarks supply, as first-class sources, exactly what the conventional artifact lacks: PaperBench (Starace et al., 2025) closes the configuration gap (8,921 expert reproduction requirements; only 45.4% fully specified in PDFs, Fig. 1c), and RE-Bench (Wijk et al., 2025) closes the exploration gap (24,008 agent runs; 90.2% of dollar cost on discarded dead ends; App. F.1). For each target, the ARA is compiled from the full source bundle via the ARA Compiler (§3.2) under the Seal Level 1 gate; the conventional baseline is the published PDF and companion GitHub repo (15/23 PaperBench papers) or, for paperless RE-Bench tasks, an LLM-synthesised paper-style writeup of the official reference solution paired with the official source. The corpus (App. F.1), the compilation pipeline, and the RE-Bench-specific beat-reference fairness filter are in App. F–I.2.

4.1. Knowledge Extraction from ARA

This layer is a precondition for the next two: a format that loses information during compilation cannot improve reproduction or extension. It targets the configuration and exploration gaps in Fig. 1c.

Setup. We probe each format with **450 questions** (15 per target × 30 targets: 23 PaperBench papers and 7 RE-Bench tasks), split into three categories: *A* (fidelity, 10/target, both benchmarks), *B* (configuration recovery, 5/PaperBench paper), and *C* (failure knowledge, 5/RE-Bench task). To avoid source bias, questions per target are drawn from two independently generated pools (one seeded from the PDF, one from the ARA), then merged and deduplicated. Each (target, format, question) triple is answered by an independent subagent with a fresh context and graded ternary (1.0/0.5/0.0) against a gold reference (templates, rubric, and judge model in App. G.1).

Results. ARA outperforms the baseline at every category and benchmark, **93.7% vs. 72.4%** (+21.3%) overall on 450 paired outcomes, and the gap decomposes along three category-specific mechanisms. On **Category A**, where the answer is in the PDF, ARA wins +14.8% overall: on the PaperBench subset where both formats can answer, ARA also consumes 12% fewer tokens (86.3K vs. 97.7K), as `PAPER.md`'s layer index turns linear document scanning into targeted file lookup; on the RE-Bench subset ARA spends more tokens (79.0K vs. 58.2K) precisely because the baseline often abandons unanswerable questions early, so the aggregate Cat A token gap (84.6K vs. 88.5K, ∼4%) understates the per-target efficiency win. On **Category B**, where papers systematically omit configuration detail, ARA wins +24.8% at comparable token usage by centralising hyperparameters and environment specs in `src/configs/`. On **Category C**, where the answer lives only in MALT trajectories, ARA wins +65.7%; the baseline has no failure source and abandons most queries with short empty answers (58K vs. 139K tokens). ARA token usage also scales with question depth (61K explicit → 96K scattered → 153K implicit-failure), while baseline usage stays flat (83K–118K), because linear PDF/repo scanning costs the same regardless of where the answer hides (App. G).

4.2. Reproduction from ARA

This layer tests whether ARA's structure translates understanding into action, targeting the configuration gap (54.6% of expert reproduction requirements missing from PDFs, Fig. 1c).

Setup. We select 15 PaperBench papers with companion GitHub repositories and curate 10 reproduction tasks per paper (**150 tasks, 1,743 rubric requirements**, stratified 50/49/51 easy/medium/hard); subtasks within each paper are ordered by difficulty so the agent builds cumulatively. Two coding agents receive the same mega-task but different source materials: the **ARA agent** sees only `PAPER.md`, `logic/`, and `src/` (no `evidence/`); the **baseline agent** sees the PDF and companion GitHub repo. Both use Claude Sonnet 4.6 with the same system prompt (paths differ only) and per-paper budgets of 14–20M tokens scaled by complexity. Expected numerical results are masked (`[X]%`) to prevent parroting. A blinded Claude Opus 4.6 judge grades every rubric requirement *yes*/*partial*/*no* without knowing the condition. The primary metric is the **difficulty-weighted success rate** (1:2:3 for easy/medium/hard), emphasising hard subtasks where structured information helps most (full task design, scoring formula, statistical tests, and per-paper analysis in Appendix H).

Results. Across all 15 papers with complete paired runs (150 subtasks, 1,743 rubric requirements), ARA achieves a difficulty-weighted success rate of **64.4%** vs. **57.4%** for the baseline; the win/tie/loss breakdown across papers is 8/5/2 (per-paper numbers in Appendix H.2). Figure 5 stratifies this aggregate by difficulty; the per-paper breakdown is deferred to Appendix H.2 (Figure 12).

Analysis. The ARA advantage grows with difficulty: easy subtasks (environment setup, model instantiation) are near-ceiling for both formats; the gap opens on medium and hard subtasks where reproduction depends on configuration content the PDF rarely supplies. Largest ARA wins (`fre`, `mechanistic-understanding`, `pinn`) come on multi-stage pipelines whose hyperparameter interactions PDFs describe only abstractly. Fabrication occurred in 2 baseline runs and 1 ARA run: structured artifacts reduce but do not eliminate hallucinated results (per-paper analysis in App. H.2).

4.3. Extension from ARA

This layer tests ARA's most ambitious claim: preserving prior failure trajectories lets the next agent extend the work more effectively. It targets the exploration gap (59.2% of tokens and 90.2% of dollar cost across 24,008 RE-Bench runs spent on dead ends the published artifact discards, Fig. 1c).

Setup. We use 5 of the 7 RE-Bench tasks (`triton_cumsum`, `restricted_mlm`, `fix_embedding`, `nanogpt_chat_rl`, `rust_codecontests`); the other two lack a usable failure-trace layer (App. I.1). Each ARA is built by a RE-Bench-specific pipeline (App. I.2) that lifts the official solution into `src/`/`logic/` and fans out one extraction sub-agent per MALT run to populate

Category	n	Acc. (%) ARA	Acc. (%) BL	Tok. (K/Q) ARA	Tok. (K/Q) BL
A: Fidelity	300	**95.6**	80.8	84.6	88.5
PaperBench	230	**96.7**	89.8	86.3	97.7
RE-Bench	70	**92.1**	51.4	79.0	58.2
B: Detail (PaperBench)	115	**92.6**	67.8	183.0	178.3
C: Failure (RE-Bench)	35	**81.4**	15.7	139.3	58.0
Overall	**450**	**93.7**	72.4	114.0	109.1

Table 1. Understanding: accuracy and per-question tokens over 450 paired outcomes. ARA wins at every category and benchmark (App. G.2).

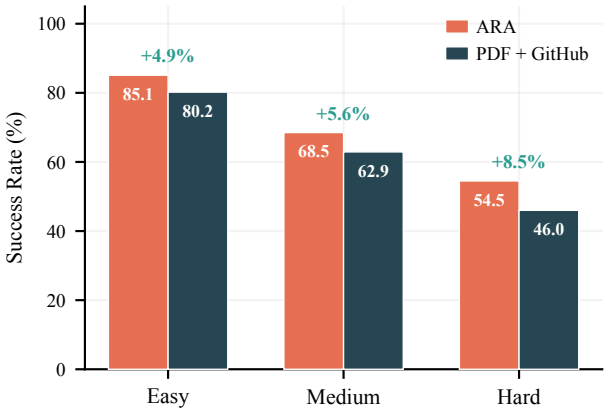


Figure 5. Reproduction success across all 15 papers, by difficulty. The ARA advantage widens monotonically.

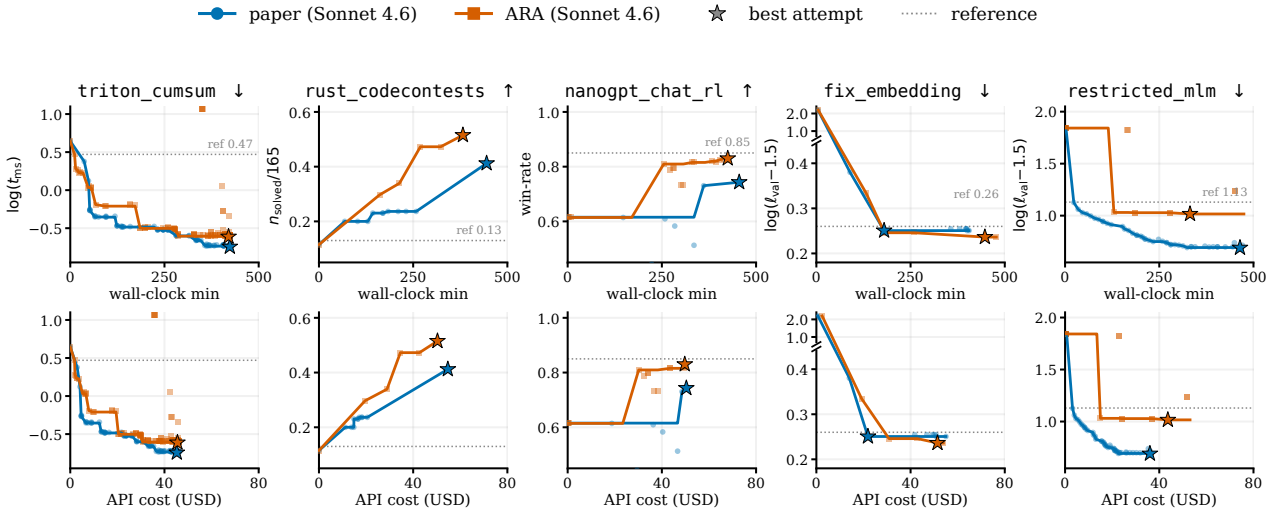


Figure 6. Extension trajectories on five RE-Bench tasks under Claude Sonnet 4.6, plotted against wall-clock time (top) and cumulative cost (bottom). *The* ARA *agent reaches a useful first move earlier on every task and ends ahead on three of five; the paper agent overtakes late on* `triton_cumsum` *and* `restricted_mlm`*, a reversal that flips back under the weaker Sonnet 4.5 base.*

`trace//evidence/`; a direction-aware *beat-reference filter* excludes any MALT attempt that exceeded the reference, the experiment's central fairness rule. Both agents start from an identical workdir (reference `solution.py`, `score.sh`, data symlinks) and differ only in the `reference/` bundle: the **paper agent** reads an LLM-synthesised paper-style writeup of the reference plus the official `src/` (the same beat-reference filter applied; construction in App. I.3), while the **ARA agent** reads the full ARA including the failure-record `trace//evidence/` layers. Both must beat the reference score by editing `solution.py` and running `bash score.sh`; we report the best score across all invocations (score-event extraction in App. I.5). Runs use the Claude Agent SDK (Anthropic, 2025b) with tools {`Bash`, `Read`, `Edit`, `Write`, `Glob`, `Grep`}, an 8 h SLURM wall clock, and a $50 API-spend cap (harness engineering and failure-mode fixes in App. I.4); all five tasks run on Claude Sonnet 4.6, with paired Sonnet 4.5 runs on `triton_cumsum` and `restricted_mlm` (App. I).

Results. Figure 6 reports the best-so-far envelope and the underlying scoring attempts per agent on each task, against wall-clock time and API spend. On `rust_codecontests`, `nanogpt_chat_rl`, and `fix_embedding` the ARA agent ends with the better best score; on `triton_cumsum` and `restricted_mlm` under Sonnet 4.6 the paper agent ends ahead. The trajectories surface three phenomena described below; per-task case studies and trace evidence (`file reads`,

ThinkingBlock reasoning, edit history) are in Appendix I.6.

Early acceleration; capability-dependent finish. The ARA agent reaches a useful first move earlier on every task: e.g., $t = 9$ min vs. 395 min for the same Rust-library idea on rust_codecontests, $t = 11$ min vs. 37 min for first scoring on triton_cumsum. Whether that lead holds depends on the base model: under Sonnet 4.6 the paper agent overtakes on triton_cumsum and restricted_mlm by discovering moves the trace never names (int8 input compression; a focused ConvMLMDilated tune); paired Sonnet 4.5 runs invert this and ARA wins both (App. Figs. 13, 14). The artifact's value scales with the gap between what the trace documents and what the agent can discover unaided; we revisit this in §M.

5. Limitations

Three limitations qualify the present work; App. L expands on each. *Scope.* Our evaluation targets machine-learning research, where computational reproducibility and discrete contribution types fit ARA's four-layer structure. Experimental sciences that hinge on physical execution, and theoretical disciplines with no meaningful Physical Layer, remain untested. *Fidelity ceiling.* A compiled ARA can preserve only what its source supplies, so legacy artifacts inherit every gap in the original PDF and repository. Closing this gap is a question of adoption, whether researchers run the Live Research Manager, rather than of further protocol design. *Deployment prerequisites.* The adversarial-robustness, privacy, and schema-migration properties claimed in §3.3 are aspirational. Sandboxed execution, anomaly detection, fine-grained access control over the Exploration Graph, and a stable migration path across major revisions are all left to future work.

6. Conclusion

The PDF was designed for human readers, and that assumption now bottlenecks an ecosystem in which agents are first-class participants in research. We propose ARA, an agent-native protocol that recasts a research contribution as a navigable, executable, and verifiable artifact rather than a static narrative. Around this core, we sketch an ecosystem, ingestion, live project management, and machine review, that lets human and machine researchers publish, reproduce, and extend scientific work on a common substrate. We view ARA as a starting point: a minimal abstraction on which an agent-native research stack can be built. We discuss limitations and broader implications in Appendices L and M.

References

Balazs Aczel, Barnabas Szaszi, and Alex O. Holcombe. A billion-dollar donation: Estimating the cost of researchers' time spent on peer review. *Research Integrity and Peer Review*, 6(14):1–8, 2021. doi: 10.1186/s41073-021-00118-2.

Anthropic. Agent skills: A simple, open format for agent capabilities. https://agentskills.io/specification, 2025a. Open specification. Accessed 2026-03-08.

Anthropic. Claude code sdk. https://docs.anthropic.com/en/docs/claude-code/sdk, 2025b. Accessed 2026-04-17.

Jinheon Baek, Sujay Kumar Jauhar, Silviu Cucerzan, and Sung Ju Hwang. ResearchAgent: Iterative research idea generation over scientific literature with large language models. In *Proceedings of NAACL-HLT*, 2025. arXiv:2404.07738.

Monya Baker. 1,500 scientists lift the lid on reproducibility. *Nature*, 533(7604):452–454, 2016. doi: 10.1038/533452a.

Vladimir Baulin, Austin Cook, Daniel Friedman, Janna Lumiruusu, Andrew Pashea, Shagor Rahman, and Benedikt Waldeck. The discovery engine: A framework for AI-driven synthesis and navigation of scientific knowledge landscapes. *arXiv preprint arXiv:2505.17500*, 2025.

Tim Baumgärtner and Iryna Gurevych. SciCoQA: Quality assurance for scientific paper–code alignment. *arXiv preprint arXiv:2601.12910*, 2026.

Lukas Biewald. Experiment tracking with Weights & Biases, 2020. URL https://www.wandb.com/. Software available from wandb.com.

Daniil A. Boiko, Robert MacKnight, Ben Kline, and Gabe Gomes. Autonomous chemical research with large language models. *Nature*, 624(7992):570–578, 2023. doi: 10.1038/s41586-023-06792-0.

A. Sina Booeshaghi, Lior Luebbert, and Lior Pachter. Science should be machine-readable. *bioRxiv*, 2026. doi: 10.64898/2026.01.30.702911.

Adam Brinckman, Kyle Chard, Niall Gaffney, Mihael Hategan, Matthew B. Jones, Kacper Kowalik, Sivakumar Kulasekaran, Bertram Ludäscher, Bryce D. Mecum, Jarek Nabrzyski, Victoria Stodden, Ian J. Taylor, Matthew J. Turk, and Kandace Turner. Computing environments for reproducibility: Capturing the "Whole Tale". *Future Generation Computer Systems*, 94:854–867, 2019. doi: 10.1016/j.future.2017.12.029.

Marco Canini. Scientists should stop writing papers for each other. LinkedIn Pulse, March 2026. Accessed 2026-03-16.

Christopher D Chambers. Registered reports: A new publishing initiative at Cortex. *Cortex*, 49(3):609–610, 2013. doi: 10.1016/j.cortex.2012.12.016.

Ziru Chen, Shijie Chen, Yuting Ning, Qianheng Zhang, Boshi Wang, Botao Yu, Yifei Li, Zeyi Liao, Chen Wei, Zitong Lu, Vishal Dey, Mingyi Xue, Frazier N. Baker, Benjamin Burns, Daniel Adu-Ampratwum, Xuhui Huang, Xia Ning, Song Gao, Yu Su, and Huan Sun. ScienceAgentBench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.05080.

Michael R. Crusoe, Sanne Abeln, Alexandru Iosup, Peter Amstutz, John Chilton, Nebojša Tijanić, Hervé Ménager, Stian Soiland-Reyes, Bogdan Gavrilović, and Carole Goble. Methods included: Standardizing computational reuse and portability with the Common Workflow Language. *Communications of the ACM*, 65(6):54–63, 2022. doi: 10.1145/3486897.

Paolo Di Tommaso, Maria Chatzou, Evan W. Floden, Pablo Prieto Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4):316–319, 2017. doi: 10.1038/nbt.3820.

Annie Franco, Neil Malhotra, and Gabor Simonovits. Publication bias in the social sciences: Unlocking the file drawer. *Science*, 345(6203):1502–1505, 2014. doi: 10.1126/science.1255484.

Luyu Gao, Zhuyun Dai, Panupong Pasupat, Anthony Chen, Arun Tejasvi Chaganty, Yicheng Fan, Vincent Y. Zhao, Ni Lao, Hongrae Lee, Da-Cheng Juan, and Kelvin Guu. RARR: Researching and revising what language models say, using language models. In *Proceedings of ACL*, pages 16477–16508, 2023.

Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. An open source AutoML benchmark. *arXiv preprint arXiv:1907.00909*, 2019.

Paul Groth, Andrew Gibson, and Jan Velterop. Anatomy of a nanopublication. *Information Services & Use*, 30(1-2): 51–56, 2010. doi: 10.3233/ISU-2010-0613.

Tianyu Hua, Harper Hua, Violet Xiang, Benjamin Klieger, Sang T. Truong, Weixin Liang, Fan-Yun Sun, and Nick Haber. ResearchCodeBench: Benchmarking LLMs on implementing novel machine learning research code. *arXiv preprint arXiv:2506.02314*, 2025.

Minghui Huang. DecMetrics: Structured claim decomposition scoring for factually consistent LLM outputs. *arXiv preprint arXiv:2509.04483*, 2025.

Mohamad Yaser Jaradeh, Allard Oelen, Kheir Eddine Farfar, Manuel Prinz, Jennifer D'Souza, Gábor Kismihók, Markus Stocker, and Sören Auer. Open research knowledge graph: Next generation infrastructure for semantic scholarly knowledge. In *Proceedings of the 10th International Conference on Knowledge Capture (K-CAP)*, pages 243–246, 2019. doi: 10.1145/3360901.3364435.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770.

Patrick Tser Jern Kon, Jiachen Liu, Xinyi Zhu, Qiuyi Ding, Jingjia Peng, Jiarong Xing, Yibo Huang, Yiming Qiu, Jayanth Srinivasa, Myungjin Lee, Mosharaf Chowdhury, Matei Zaharia, and Ang Chen. EXP-Bench: Can AI conduct AI research experiments? *arXiv preprint arXiv:2505.24785*, 2025.

Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012. doi: 10.1093/bioinformatics/bts480.

Thomas S. Kuhn. *The Structure of Scientific Revolutions*. University of Chicago Press, Chicago, 1962.

Keigo Kusumegi, Xinyu Yang, Paul Ginsparg, Mathijs de Vaan, Toby Stuart, and Yian Yin. Scientific production in the era of large language models. *Science*, 390(6779): 1240–1243, 2025. doi: 10.1126/science.adw3000.

Timothy Lebo, Satya Sahoo, Deborah McGuinness, Khalid Belhajjame, James Cheney, David Corsar, Daniel Garijo, Stian Soiland-Reyes, Stephan Zednik, and Jun Zhao. PROV-O: The PROV ontology. W3c recommendation, W3C, 2013. URL https://www.w3.org/TR/prov-o/.

Lehui Li, Ruining Wang, Haochen Song, Yaoxin Mao, Tong Zhang, Yuyao Wang, Jiayi Fan, Yitong Zhang, Jieping Ye, Chengqi Zhang, and Yongshun Gong. What papers don't tell you: Recovering tacit knowledge for automated paper reproduction. *arXiv preprint arXiv:2603.01801*, 2026.

Amber Liu. The rise of AI-native researchers. https://amberliu2.substack.com/p/the-rise-of-ai-native-researchers, 2026a. Blog post. Accessed 2026-03-08.

Jiachen Liu, Maestro Harmon, and Zechen Zhang. Scireasoning: A dataset decoding AI innovation patterns. *arXiv preprint arXiv:2601.04577*, 2026.

Ziming Liu. Research agents should target knowledge graphs, not papers. https://kindxiaoming.github.io/blog/2026/research-agent/, 2026b. Blog post. Accessed 2026-03-08.

Chris Lu, Cong Lu, Robert Tjarko Lange, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024.

Yujie Luo, Zhuoyun Yu, Xuehai Wang, Yuqi Zhu, Ningyu Zhang, Lanning Wei, Lun Du, Da Zheng, and Huajun Chen. What makes AI research replicable? Executable knowledge graphs as scientific knowledge representations. *arXiv preprint arXiv:2510.17795*, 2025.

Andres M. Bran, Sam Cox, Oliver Schilter, Carlo Baldassari, Andrew D. White, and Philippe Schwaller. Augmenting large language models with chemistry tools. *Nature Machine Intelligence*, 6(5):525–535, 2024. doi: 10.1038/s42256-024-00832-8.

Natalie Matosin, Elisabeth Frank, Martin Engel, Jeremy S Lum, and Kelly A Newell. Negativity towards negative results: a discussion of the disconnect between scientific worth and scientific culture. *Disease Models & Mechanisms*, 7(2):171–173, 2014. doi: 10.1242/dmm.015123.

Peter B. Medawar. Is the scientific paper a fraud? *The Listener*, 70:377–378, 1963. Reprinted in *The Strange Case of the Spotted Mice*, Oxford University Press, 1996.

OpenAI. AGENTS.md: A standard for agent-oriented repository documentation. https://github.com/openai/agents.md, 2025. Accessed 2026-03-01.

Joelle Pineau, Philippe Vincent-Lamarre, Koustuv Sinha, Vincent Larivière, Alina Beygelzimer, Florence d'Alché Buc, Emily Fox, and Hugo Larochelle. Improving reproducibility in machine learning research: A report from the NeurIPS 2019 reproducibility program. *Journal of Machine Learning Research*, 22(164):1–20, 2021.

Sebastian Pineda Arango, Hadi S. Jomaa, Martin Wistuba, and Josif Grabocka. HPO-B: A large-scale reproducible benchmark for black-box HPO based on OpenML. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, 2021. URL https://arxiv.org/abs/2106.06257.

Michael Polanyi. *The Tacit Dimension*. Doubleday, Garden City, NY, 1966.

Petar Radanliev, Omar Santos, Carsten Maple, and Sepideh Atefi. Operationalising artificial intelligence bills of materials for verifiable AI provenance and lifecycle assurance. *Frontiers in Computer Science*, 8:1735919, 2026. doi: 10.3389/fcomp.2026.1735919.

Razeen A Rasheed, Somnath Banerjee, Animesh Mukherjee, and Rima Hazra. From fluent to verifiable: Claim-level auditability for deep research agents. *arXiv preprint arXiv:2602.13855*, 2026.

Allen H. Renear and Carole L. Palmer. Strategic reading, ontologies, and the future of scientific publishing. *Science*, 325(5942):828–832, 2009. doi: 10.1126/science.1157784.

Robert Rosenthal. The file drawer problem and tolerance for null results. *Psychological Bulletin*, 86(3):638–641, 1979. doi: 10.1037/0033-2909.86.3.638.

Adam Rule, Aurélien Tabard, and James D Hollan. Exploration and explanation in computational notebooks. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–12, 2018. doi: 10.1145/3173574.3173606.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. Agent laboratory: Using LLM agents as research assistants. *arXiv preprint arXiv:2501.04227*, 2025.

Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. Paper2Code: Automating code generation from scientific papers in machine learning. *arXiv preprint arXiv:2504.17192*, 2025. ICLR 2026.

Stian Soiland-Reyes, Peter Sefton, Mercè Crosas, Leyla Jael Castro, Frederik Coppens, José M Fernández, Daniel Garijo, Björn Grüning, Marco La Rosa, Simone Leo, et al. Packaging research artefacts with RO-Crate. *Data Science*, 5(2):97–138, 2022. doi: 10.3233/DS-210053.

Giulio Starace, Oliver Jaffe, Dane Sherburn, James Aung, Jun Shern Chan, Leon Maksin, Rachel Dias, Evan Mays, Benjamin Kinsella, Wyatt Thompson, Johannes Heidecke, Amelia Glaese, and Tejal Patwardhan. PaperBench: Evaluating AI's ability to replicate AI research. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267, pages 56843–56873. PMLR, 2025. URL https://arxiv.org/abs/2504.01848.

Markus Stocker, Mark Snyder, Chiara Anfuso, Marian Ludwig, et al. Rethinking the production and publication of machine-readable expressions of research findings. *Scientific Data*, 12(1):1–10, 2025. doi: 10.1038/s41597-025-04905-0.

Victoria Stodden, Marcia McNutt, David H Bailey, Ewa Deelman, Yolanda Gil, Brooks Hanson, Michael A Heroux, John PA Ioannidis, and Michela Taufer. Enhancing reproducibility for computational methods. *Science*, 354 (6317):1240–1241, 2016. doi: 10.1126/science.aah6168.

Aristidis Vasilopoulos. Codified context: Infrastructure for AI agents in a complex codebase. *arXiv preprint arXiv:2602.20478*, 2026.

David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. Fact or fiction: Verifying scientific claims. In *Proceedings of EMNLP*, pages 7534–7550, 2020. doi: 10.18653/v1/2020.emnlp-main.609.

Fiona Y. Wang, Lee Marom, Subhadeep Pal, Rachel K. Luu, Wei Lu, Jaime A. Berkovich, and Markus J. Buehler. Autonomous agents coordinating distributed discovery through emergent artifact exchange. *arXiv preprint arXiv:2603.14312*, 2026.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Hjalmar Wijk, Tao Lin, Joel Becker, Sami Jawhar, Neev Parikh, Thomas Broadley, Lawrence Chan, Michael Chen, Josh Clymer, Jai Dhyani, Elena Ericheva, Katharyn Garcia, Brian Goodrich, Nikola Jurkovic, Holden Karnofsky, Megan Kinniment, Aron Lajko, Seraphina Nix, Lucas Sato, William Saunders, Maksym Taran, Ben West, and Elizabeth Barnes. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents against human experts. In *Proceedings of the 42nd International Conference on Machine Learning*, 2025. URL https://arxiv.org/abs/2411.15114.

Mark D Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Mick Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E Bourne, et al. The FAIR guiding principles for scientific data management and stewardship. *Scientific Data*, 3(1):1–9, 2016. doi: 10.1038/sdata.2016.18.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *Conference on Language Modeling (COLM)*, 2024. arXiv:2308.08155.

Yutaro Yamada, Robert Tjarko Lange, Cong Lu, Shengran Hu, Chris Lu, Jakob Foerster, Jeff Clune, and David Ha. The AI scientist-v2: Workshop-level automated scientific discovery via agentic tree search. *arXiv preprint arXiv:2504.08066*, 2025.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.

Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. NAS-Bench-101: Towards reproducible neural architecture search. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 7105–7114. PMLR, 2019.

Matei Zaharia, Andrew Chen, Aaron Davidson, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, Siddharth Murching, Tomas Nykodym, Paul Ogilvie, Mani Parkhe, et al. MLflow: A system for managing the machine learning lifecycle. 2018. Workshop on ML Systems at NeurIPS.

Guibin Zhang, Junhao Wang, Junjie Chen, Wangchunshu Zhou, Kun Wang, and Shuicheng Yan. AgenTracer: Who is inducing failure in the LLM agentic systems? *arXiv preprint arXiv:2509.03312*, 2025.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems (NeurIPS), Datasets and Benchmarks Track*, 2023.

Kunlun Zhu, Zijia Liu, Bingxuan Li, Muxin Tian, Yingxuan Yang, Jiaxun Zhang, Pengrui Han, Qipeng Xie, Fuyang Cui, Weijia Zhang, Xiaoteng Ma, Xiaodong Yu, Gowtham Ramesh, Jialian Wu, Zicheng Liu, Pan Lu, James Zou, and Jiaxuan You. Where LLM agents fail and how they can learn from failures. *arXiv preprint arXiv:2509.25370*, 2025.

A. ARA Protocol Details

This appendix consolidates the empirical motivation and the protocol specification for the ARA format. Sections A.1–A.3 quantify the two structural costs of narrative compilation that ARA is designed to eliminate (Engineering Tax in App. A.2, Storytelling Tax in App. A.3; cf. §1). Sections A.4–A.5 document the protocol itself: the kernel/repository modes for the Physical Layer and a worked example using this paper's own artifact.

A.1. Taxonomy of Reproduction-Critical Information

To understand *what* information an agent needs to reproduce a paper—and where PDFs fall short—we analyze the expert-authored rubrics from PaperBench (Starace et al., 2025), a benchmark that evaluates AI agents on full paper reproduction. Each rubric decomposes a paper into atomic *leaf requirements*: individually verifiable conditions that collectively constitute a faithful reproduction. **Scope.** The taxonomy below is derived from a deeply annotated 5-paper subset (3,050 leaves), chosen for tractability of fine-grained category labeling; the coarser per-task-category and gap-type frequencies reported in §4.1 and Appendix A.2 are validated on the full 23-paper corpus (8,921 requirements). The subset spans diverse domains—black-box LLM adaptation (BBox), mechanistic interpretability, continual RL (Self-Composing Policies), physics-informed neural networks (PINN), and foundation models for RL (FRE).

By categorizing every leaf into a taxonomy of information types, we reveal both the *diversity* of knowledge needed for reproduction and the specific failure modes that arise when this knowledge is scattered across a narrative PDF rather than organized in a structured artifact.

A.1.1. Information Categories

We identify ten categories of reproduction-critical information. Table 2 summarizes the categories with their frequency distribution across our analyzed rubrics.

Below we define each category, give concrete examples drawn from the PaperBench rubrics, and explain how ARA's structure addresses the underlying retrieval challenge.

1. Combinatorial experiment matrix (24.1%). The single largest category consists of requirements that enumerate which model variant must be run on which dataset, with which configuration, for how many seeds. In PDFs, this combinatorial structure is compressed into a single sentence ("We evaluate all methods on three task sequences with 10 seeds each") or a results table whose row/column headers implicitly define the cross-product. An agent must mentally decompose the matrix to know, e.g., that "CompoNet on Freeway, 10 seeds, 1M timesteps per task" is a distinct run.

Examples:

- *self-composing-policies*: 62 leaves enumerate {6 methods} × {3 task sequences} × {seeds, timesteps, trained}—each a separate verifiable requirement (e.g., "CompoNet on Meta-World: 10 seeds, 1M timesteps/task, trained").
- *bbox*: 46 evaluation leaves cross {5 model sizes} × {4 datasets} × {3 feedback types} × {single-step, full-step} inference modes.
- *pinn*: ∼1,273 leaves enumerate a grid of {4 PDE problems} × {4 network widths} × {5 learning rates} × {3 optimizers}, each combination a distinct training run.

ARA advantage. The `experiments.md` file makes every cell of the experiment matrix explicit, with structured `Setup` fields that list model, dataset, and configuration as machine-readable key–value pairs. An agent can enumerate all runs programmatically rather than parsing table headers.

2. Evaluation protocol (18.5%). Requirements specifying *which* metric to compute, on *which* test split, using *which* evaluation-time configuration (e.g., beam size, number of evaluation episodes, specific layers to probe). These details are often scattered: the metric definition appears in §3, the test split in §4, the evaluation episodes in the appendix, and the layer indices in a figure caption.

Examples:

- *mechanistic-understanding*: "Compute cosine similarity between δ_i and $\delta_{\text{mlp},i}$ for layers 0, 2, 4, 6, 8, 10, 12, 14, 16, 18 using 1,199 prompts from RealToxicityPrompts."
- *fre*: "Evaluation is repeated and averaged over 20 episodes and 5 seeds; 32 state-reward pairs are sampled from the evaluation task environment."
- *bbox*: "The Chain-of-Thought baseline has been evaluated on the test splits of all datasets using GPT-3.5 Turbo."

ARA advantage. Each experiment entry in `experiments.md` has a declarative `Procedure` field that specifies evaluation steps as an ordered list, and a `Metrics` field that names the exact metrics. The `configs/training.md` file separately records evaluation-time parameters (e.g., beam size, episodes).

3. Hyperparameters (17.2%). Classic training configuration: learning rates, batch sizes, optimizer parameters, temperature, weight decay, LoRA rank, number of epochs. While these are the *most commonly discussed* reproduction barrier, they account for only 17% of leaf requirements. In PDFs, hyperparameters are typically consolidated in an appendix table, but the correspondence between table rows and specific experimental conditions is often ambiguous.

Examples:

- *bbox*: "AdamW optimizer with learning rate 5e-6, weight decay 0.01; batch size 64; 6,000 training steps."
- *self-composing-policies*: 29 leaves enumerate every SAC and PPO parameter individually (e.g., "SAC: target smoothing coefficient $\tau = 0.005$"; "PPO: GAE $\lambda = 0.95$").
- *pinn*: "Learning rate of the Adam optimizer can be set to 1E-5, 1E-4, 1E-3, 1E-2, or 1E-1."

Category	%	PDF Difficulty	ARA Layer
Combinatorial experiment matrix	24.1	Implicit in prose	`experiments.md`
Evaluation protocol	18.5	Scattered across §, appendix	`experiments.md`
Hyperparameters	17.2	Buried in appendix tables	`configs/training.md`
Metric logging	10.4	Rarely specified	`experiments.md`
Result interpretation	8.6	Mixed with discussion	`claims.md`, `evidence/`
Architecture specification	5.8	Split across text, figures, appendix	`architecture.md`
Mathematical formulation	4.5	Equation references break across sections	`algorithm.md`
Implementation tricks	4.2	Footnotes, appendix asides	`heuristics.md`
Data pipeline	3.8	Preprocessing details omitted	`configs/`, `environment.md`
Environment / infrastructure	2.9	Assumed known	`environment.md`

Table 2. Taxonomy of reproduction-critical information in PaperBench rubrics. Frequency is computed across 3,050 leaf requirements from five papers. The "PDF Difficulty" column characterizes the primary challenge of extracting this information from a narrative PDF. The "ARA Layer" column identifies which ARA component directly addresses each category.

ARA advantage. The `configs/training.md` file provides a single, authoritative location for all hyperparameters, organized by experiment. The `heuristics.md` file additionally records *sensitivity* annotations (low/medium/high) and valid *bounds*, information that PDFs almost never provide.

4. Metric computation and logging (10.4%). Requirements that the agent must *record* specific intermediate quantities during runs: loss curves, attention distributions, cost tracking (dollars per 1k questions), episodic returns logged every N steps. This "instrumentation" knowledge is rarely specified in papers—authors implicitly know what to log but do not document it as part of the method.

Examples:

- *bbox*: 71 leaves (25% of the paper's rubric) require computing and saving training costs, inference costs (USD/1k questions), and evaluation costs across all dataset $\times$ variant combinations.
- *self-composing-policies*: "Output attention distribution logged every 10k timesteps"; "Matching rate between final output and internal policy, saved every 10k steps."

ARA advantage. The `experiments.md` `Metrics` and `Procedure` fields can explicitly list what to log and at what frequency. The `evidence/` layer provides concrete examples of the expected output format.

5. Result interpretation (8.6%). Qualitative claims about what the results should *show*—directional trends, comparative rankings, mechanistic explanations. These carry the highest weight in PaperBench rubrics (weight = 2) because they test whether the agent *understands* the results, not just whether the code ran.

Examples:

- *mechanistic-understanding*: "After adapting with DPO, the principal component of the residual streams shift in the same direction, and the activation of the toxic vectors decrease." (Weight = 2)
- *mechanistic-understanding*: "The extracted tokens encode different characteristics of toxic language: tokens from $\mathbf{W}$ are mostly curse words; MLP.vToxic are a mix of curse words and insults; SVD.uToxic encode insults and female sexual references." (Weight = 2)
- *self-composing-policies*: "CompoNet achieves higher average performance and forward transfer than all baselines on all three task sequences."
- *pinn*: "Adam+L-BFGS always achieves the lowest minimum loss compared to just using Adam or L-BFGS alone."

ARA advantage. The `claims.md` file states each claim with explicit `Falsification criteria` and pointers to the experiment that verifies it. The `experiments.md` `Expected outcome` field records the directional prediction (e.g., "method A outperforms method B") without revealing exact numbers, enabling blind reproduction.

6. Architecture specification (5.8%). Layer counts, channel sizes, activation functions, output head structure, embedding dimensions. In PDFs, architecture details are split across a figure (showing the high-level diagram), the methods section (describing components in prose), and the appendix (listing dimensions in a table). An agent must men-

tally compose these three sources to build the full specification.

Examples:

- *self-composing-policies*: "CNN has three convolutional layers with 32, 64, and 64 channels and filter sizes of 8, 4, and 3"; "SAC: hidden dimension $d_{\text{model}} = 256$; critic network has 3 layers; activation is ReLU."
- *fre*: "GC-BC model is a MLP with three hidden layers of size 512"; "layer normalization is applied before each activation function."
- *mechanistic-understanding*: "Binary classifier of the form $\text{softmax}(\mathbf{W}\mathbf{x})$ where $\mathbf{W}$ has dimensionality $K \times 2$."

ARA advantage. The `architecture.md` file provides a single location listing every component with its dimensions, activation functions, and input/output specifications. Code stubs in `src/code/` provide an executable complement.

7. Mathematical formulation (4.5%). Specific equations that must be implemented exactly: loss functions, attention operations, PDE boundary conditions, update rules. In PDFs, equations are referenced by number, but the reader must trace through variable definitions scattered across multiple sections.

Examples:

- *self-composing-policies*: "Output attention: $\text{softmax}(qK^T/\sqrt{d_{\text{model}}}) \cdot V$"; "Forward transfer: $\text{FTr}_i = (\text{AUC}_i - \text{AUC}_i^b)/(1 - \text{AUC}_i^b)$."
- *pinn*: "The loss function corresponds to the non-linear least squares problem described in Section 2.1, with the relevant differential operator and boundary/initial condition operators outlined in Appendix A.1."
- *fre*: "The value function is updated with an expectile regression objective on the critic's Q-values"; "The actor is updated via advantage-weighted regression (AWR)."

ARA advantage. The `algorithm.md` file presents the algorithm as a self-contained pseudocode block with all variable definitions in scope. The `concepts.md` file defines notation and links to the equations that use each symbol.

8. Implementation tricks (4.2%). Non-obvious design choices that distinguish faithful reproduction from naive re-implementation: weight freezing schedules, initialization from prior checkpoints, gradient clipping thresholds, normalization details, optimizer switching strategies. These are the hardest items to recover from a PDF because they appear as parenthetical remarks, footnotes, or single sentences buried in dense paragraphs.

Examples:

- *self-composing-policies*: "Single CNN encoder per policy; new encoder initialized with weights of the previous one" (Appendix E.2); "Reset critic network at the beginning of each task"; "Normalize summed vectors for continuous action spaces."
- *fre*: "The transformer does not use a causal mask on its attention"; "Positional embeddings are not used in the transformer"; "States sampled for decoding and encoding are sampled separately."
- *pinn*: "At the end of training, the L-BFGS directions, steps, and inverse of inner products are saved" (Appendix C.2); "Strong Wolfe line search is used with L-BFGS."

ARA advantage. The `heuristics.md` file is *specifically designed* to capture these items. Each heuristic entry includes `Rationale` (why it matters), `Sensitivity` (how much performance degrades without it), and `Code ref` (where in the code to apply it).

9. Data pipeline (3.8%). Dataset acquisition, split ratios, filtering criteria, preprocessing steps, data augmentation, collocation point sampling strategies. These details are often under-specified in papers ("we use the standard train/test split") or tucked into a single appendix paragraph.

Examples:

- *bbox*: "Split GSM8K into 7,473 training and 1,319 test samples"; "Randomly sample 100 questions for TruthfulQA test set, remaining 717 for training."
- *mechanistic-understanding*: "24,576 pairs of toxic and non-toxic continuations have been created"; "295 prompts selected from RealToxicityPrompts that output 'shit' as the next token."
- *pinn*: "10,000 residual points randomly sampled from a 255×100 grid; 257 equally spaced points for each initial condition and 101 for each boundary condition."

ARA advantage. The `configs/` directory provides structured configuration files with exact split sizes and sampling parameters. The `environment.md` file specifies dataset versions and download URLs.

10. Environment and infrastructure (2.9%). Specific API endpoints, model version strings, library versions, simulator names, hardware requirements. These are often assumed to be "obvious" and omitted entirely from the paper, yet they are essential for reproduction.

Examples:

- *bbox*: "API access configured for davinci-002"; "Code to execute fine-tuning jobs through the Azure OpenAI API"; "Mixtral-8x7B-v0.1 loaded from HuggingFace in half-precision."
- *self-composing-policies*: 15 leaves enumerate specific Gymnasium environment IDs (e.g., `ALE/SpaceInvaders-v5`, `hammer-v2`) and required packages (`Metaworld` from Farama-Foundation).
- *fre*: "The observation space's XY coordinates are discretized into 32 bins for Ant Maze agents."

ARA advantage. The `environment.md` file lists exact package versions, model identifiers, and hardware requirements. The `configs/model.md` file records model names, sizes, and loading configurations (e.g., precision, quantization).

A.1.2. KEY FINDINGS

Three observations emerge from this analysis:

Hyperparameters are necessary but not sufficient. Classic hyperparameters—the most discussed reproduction barrier—account for only 17.2% of leaf requirements. The remaining 82.8% comprise evaluation protocols, experiment matrices, logging requirements, result interpretation targets, and implementation tricks that are harder to locate in a PDF and receive less attention in reproducibility discussions.

The combinatorial explosion is the dominant challenge. The largest category (24.1%) consists of requirements that enumerate the full cross-product of models, datasets, and configurations. In a PDF, this matrix is compressed into a single table or sentence; an agent must decompose it into individual runs. The ARA format makes this matrix explicit and machine-enumerable.

High-weight requirements demand *understanding*, not just extraction. All weight-2 requirements in the PaperBench rubrics belong to the "Result Interpretation" category. These test whether the agent can verify that reproduced results exhibit the *qualitative patterns* claimed by the paper—not just whether the code runs. The ARA `claims.md` and `experiments.md` layers directly encode these verification targets, making the connection between code output and paper claims explicit rather than requiring the agent to re-derive it from narrative text.

A.2. PDF Information Gap Distribution

While §A.1 characterises *what* information reproduction requires, this subsection asks the dual question: of those requirements, which are actually specified in the source PDF? The two views are complementary; together they show that the gap is not in any one category but is structural across the document.

Methodology. PaperBench's expert reproduction rubric for each paper enumerates the requirements an agent must satisfy to reproduce the paper's results. For each of the 8,921 leaf requirements across the 23 papers, we compare the requirement against the source PDF and label it *sufficient* (the PDF text fully specifies what is needed), *partial* (some components specified but key details missing), or *absent* (the PDF does not address the requirement); each label is accompanied by an annotator confidence rating (high / medium / low). When the label is partial or absent, we additionally tag the requirement with a gap-type category (missing hyperparameter, vague description, cross-reference-only, etc.). Labels are produced by an LLM-as-judge run per (requirement, PDF excerpt) pair, with the judge required to cite the PDF passage that supports its decision; the headline 45.4% sufficient figure is dominated by the 64% high-confidence subset (full pipeline in `code/eval/pdf_information_gap.py`).

Per-category coverage. Table 3 breaks the 8,921 requirements down by PaperBench task category. The median paper shows 45.3% sufficient and 47.9% partial, confirming the gap is systemic rather than driven by outliers.

Task Category	Reqs	Sufficient	Partial	Absent
Code Development	3,942	37.3%	54.9%	7.8%
Code Execution	4,355	50.5%	47.9%	1.6%
Result Analysis	624	60.6%	36.9%	2.6%
Overall	**8,921**	**45.4%**	**50.2%**	**4.4%**

Table 3. Reproduction information gap across 23 PaperBench papers (8,921 requirements). PDFs systematically under-specify the information needed for reproduction, with the largest gaps in code development and dataset acquisition.

Gap-type breakdown. Table 4 breaks down the 8,921 reproduction requirements by gap type. The three largest categories—missing hyperparameters (26.2%), vague descriptions (21.9%), and cross-reference-only specifications (13.4%)—account for over 60% of all gaps and are precisely the information types that structured formats address by design. At the fine-grained level, Dataset Acquisition achieves only 5.4% sufficient coverage (25.5% entirely absent)—no paper in the corpus consistently provides download URLs, preprocessing scripts, or data format specifications. Evaluation, Metrics & Benchmarking sits at 30.0%: papers often state *which* metrics they use but not *how* they compute them (binning strategies, confidence intervals, statistical tests).

Gap Type	Count	% of Gaps
Missing hyperparameter	2,558	26.2%
Vague description	2,141	21.9%
Cross-reference only	1,313	13.4%
Missing code detail	1,064	10.9%
Missing baseline detail	1,054	10.8%
Missing URL	538	5.5%
Figure only	499	5.1%
Ambiguous specification	399	4.1%
Implicit assumption	150	1.5%

Table 4. Distribution of information gap types across 8,921 requirements. The three largest categories—missing hyperparameters, vague descriptions, and cross-references—are precisely the gaps that structured formats address by design.

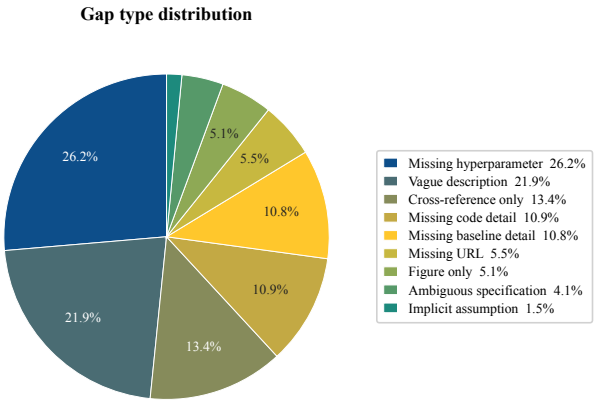


Figure 7. The three largest gap types are precisely the categories ARA's structured layers address.

Cross-view alignment. Reading Tables 2 and 4 together makes the failure mode concrete: the heaviest reproduction need (Combinatorial experiment matrix, 24.1%) is also among the most poorly specified (Vague description, 21.9% of gaps), and the third-largest need (Hyperparameters, 17.2%) maps almost one-to-one onto the largest gap (Missing hyperparameter, 26.2%). The two analyses agree on which categories ARA's structured layers must rescue.

A.3. Exploration Cost Distribution

App. A.2 quantifies the Engineering Tax (§1); this subsection quantifies its dual, the Storytelling Tax, by measuring the cost of exploration trajectories that narrative compilation discards.

What this measures (and what it is not). Across 24,008 agent runs (21 frontier models, 228 tasks) in the METR MALT corpus, 59.2% of tokens and 90.2% of dollar cost ($63,483 total) are spent in runs that did not reach the task's reference score. This is not wasted research effort: those runs map dead ends, rule out alternatives, and narrow the strategy space the next agent should consider. The cost only *becomes* waste downstream, when the next agent does not have access to that exploration and must rediscover the same dead ends from scratch. The exploration tax we report is therefore the per-agent cost of rediscovery if the failure record is not propagated, not a property of any individual run.

Breakdown. Table 5 gives the per-run breakdown. The mean below-reference token cost is 8.6× the cost of a reference-reaching run (2.58 M vs. 300 K tokens per run), with a median of 113×. Within the 59.2% of tokens that do not reach reference, 44.8% are spent in runs that produce no measurable improvement and 14.4% in runs that re-derive solutions other agents had already produced. The pattern concentrates where research-like work happens: RE-Bench tasks (the most open-ended) end below reference 73.4% of the time, vs. 47.0% on moderate-difficulty HCAST tasks and 0.7% on well-defined SWAA tasks. At the per-task level, easy tasks reach reference 85.4% of the time, medium 30.7%, and hard only 15.1%.

Metric	Tokens	Cost
Below-ref. rate (overall)	31.6%	
Below-ref. rate (RE-Bench)	73.4%	
Cost ratio (median)	113×	
Dead-end exploration	44.8%	—
Re-derivation	14.4%	—
Total below-reference	**59.2%**	**90.2%**

Table 5. Cost of below-reference exploration across 24,008 agent runs (21 frontier models, 228 tasks). The exploration itself is necessary research work; the cost only becomes waste when subsequent agents must re-incur it because the failure record is not preserved.

A.4. Physical Layer Modes: Kernel vs Repository

The Physical Layer (`/src`) adopts one of two modes, declared in the `PAPER.md` frontmatter (`src_mode: kernel | repo`) so that consuming agents adapt their strategy immediately. These two modes cover the dominant contribution types in empirical CS, where executable code is the natural physical representation.

Kernel mode (`/src/kernel/`). When the contribution is primarily *algorithmic*, the invariant can be cleanly separated from scaffolding. The kernel contains only the core modules with typed I/O signatures—often one to two orders of magnitude smaller than the full repository—stripped of all environment-specific code. A general-purpose coding agent consumes the kernel alongside the structured

specification in `/logic/solution/` and generates fresh, environment-native boilerplate in minutes. Because agent coding capabilities improve continuously, the same kernel yields a *better* surrounding implementation over time: the artifact appreciates rather than decays.

Repository mode (`/src/repo/`). When the contribution is primarily *systemic*—a CUDA kernel, a distributed training strategy, a systems architecture—the engineering *is* the contribution. The full implementation is retained but *annotated*: an `index.md` manifest maps each source file to the ARA component it implements—which claim it supports, which heuristic it embodies, which architectural module it belongs to—providing the structured navigation that a monolithic codebase lacks. Forensic bindings connect code regions to claims, constraints, and heuristics, so an agent traverses the codebase guided by research structure rather than by directory conventions alone.

In both modes, the Cognitive Layer remains the primary interface for understanding the contribution; the Physical Layer provides executable evidence, scaled to match.

A.5. ARA by Example: This Paper's Own Artifact

This paper is itself maintained as an ARA artifact. The `ara/` directory at the repository root contains the living cognitive, physical, and exploration layers that were populated incrementally during the research process (see §3.1). We reproduce excerpts from each key file below to give readers a concrete sense of the format. All entries are real; only trailing items are elided for space.

A.5.1. Directory layout and root manifest

The complete `ara/` directory for this paper is shown below. An agent's first action is to read `PAPER.md`, which contains YAML frontmatter and an abstract (∼500 tokens) sufficient to decide relevance without loading any layer.

```
ara/
  PAPER.md # entry point
  logic/ # Cognitive Layer
    problem.md # observations, gaps, key insight
    claims.md # 16 falsifiable claims + status
    experiments.md # verification plan (E1-E6)
    related_work.md # typed citation dependencies
    solution/
      heuristics.md # 23 design decisions + rationale
  trace/ # Exploration Graph
    exploration_tree.yaml # 114-node decision DAG
    sessions/ # 38 session logs (2026-03-12..04-26)
      session_index.yaml # chronological index
      2026-03-12_001.yaml # ...one file per session
    pm_reasoning_log.yaml # Live PM reasoning trace
  evidence/ # Evidence Layer
    README.md # index of raw results
  staging/
    observations.yaml # 94 unpromoted observations
```

The root manifest `PAPER.md`:

```
---
title: "Agent-Native Research Artifacts"
authors: ["Anonymous", "Anonymous"]
venue: "Anonymous Venue"
status: draft
date_created: "2026-03-12"
last_updated: "2026-04-27"
abstract: >
  We propose the Agent-Native Research Artifact
  (ARA), a file-system protocol that replaces the
  narrative paper with a machine-executable research
  package organized across four interlocking layers:
  a Cognitive Layer (/logic) encoding structured
  scientific reasoning, a Physical Layer (/src)
  containing the executable code kernel, an
  Exploration Graph (/trace) preserving the full
  branching research trajectory including dead ends,
  and an Evidence Layer (/evidence) grounding every
  claim in raw empirical results. PDF publication
  imposes two structural costs on autonomous
  research: a Storytelling Tax (failed experiments
  and rejected hypotheses are discarded to fit a
  linear narrative) and an Engineering Tax (the gap
  between reviewer-sufficient prose and
  agent-sufficient specification leaves critical
  implementation details unwritten). On PaperBench
  and RE-Bench, ARA raises question-answering
  accuracy from 72.4% to 93.7% and reproduction
  success from 57.4% to 64.4%; on RE-Bench's five
  open-ended extension tasks, the failure traces
  preserved in ARA accelerate research progress by
  helping the agent avoid pitfalls prior runs
  already mapped, but for a sufficiently capable
  model the same recorded playbook can constrain a
  more creative agent that would otherwise step
  outside the prior-run box.
layers:
  logic: logic/
  src: src/
  trace: trace/
  evidence: evidence/
  staging: staging/
---

# Layer Index

- **Cognitive** (`logic/`): structured reasoning
  - `problem.md` -- observations, gaps, key insight
  - `claims.md` -- 16 falsifiable claims with status
      and proof pointers
  - `experiments.md` -- verification plan (E1-E6)
  - `related_work.md` -- typed citation dependency graph
  - `solution/heuristics.md` -- 23 design decisions
      with rationale and sensitivity
- **Exploration** (`trace/`): branching trajectory
  - `exploration_tree.yaml` -- 114-node decision DAG
  - `sessions/` -- 38 session logs (2026-03-12..04-26)
  - `pm_reasoning_log.yaml` -- Live PM reasoning trace
- **Evidence** (`evidence/`): raw empirical results
  - `README.md` -- index of all evaluation data,
      including post-paper RE-Bench extension evals
- **Staging** (`staging/`): unpromoted observations
  - `observations.yaml` -- 94 preliminary observations
      (latest: 2026-04-26 cross-model and synthesis)
```

A.5.2. Cognitive Layer: `logic/claims.md`

Each claim carries a machine-readable status (`hypothesis`, `supported`, `testing`), falsification criteria, and proof pointers that reference the evidence layer rather than inlining results. We show two claims at different lifecycle stages.

```
# Claims

## C04: Universal Ingestor produces lossless
      transformations
```

```
- **Statement**: The LLM-based Ingestor faithfully
    transforms PDF papers into ARA format without
    information loss, achieving near-parity on
    factual Q&A between ARA and source PDF.
- **Status**: supported
- **Provenance**: ai-executed
- **Falsification criteria**: Systematic accuracy
    drop (>5%) on understanding questions.
- **Proof**: [evidence/README.md ->
    understanding_eval; 450 Qs across Cat A/B/C]
- **Dependencies**: [C03]
- **Tags**: ingestor, fidelity

## C06: Negative knowledge is the highest-value
       signal
- **Statement**: The Exploration Graph's dead-end
    documentation produces the largest accuracy gap
    in the entire evaluation -- agents with failure
    traces answer questions about failed approaches
    that narrative formats make structurally
    unanswerable.
- **Status**: supported
- **Provenance**: ai-executed
- **Falsification criteria**: Dead-end docs produce
    no measurable improvement on Cat C; gap <10pp.
- **Proof**: [evidence/README.md ->
    understanding_eval Cat C;
    evidence/README.md -> extension_eval]
- **Dependencies**: [C05]
- **Tags**: exploration-graph, negative-knowledge
... <!-- 16 claims total -->
```

A.5.3. COGNITIVE LAYER: LOGIC/PROBLEM.MD

The problem file decomposes the motivation into typed observations (empirical facts), gaps (what existing approaches miss), and a key insight that bridges them. Each entry carries evidence pointers and implication fields, so an agent can trace the full argumentative chain without reading the paper's introduction. We show one representative entry per section.

```
# Problem

## Observations
### O3: Frontier LLMs fail on research implementation
- **Statement**: Even the strongest frontier LLMs
    correctly implement fewer than 40% of novel
    research contributions when given the full paper
    and codebase, with semantic misalignment as the
    dominant failure mode.
- **Evidence**: ResearchCodeBench (Hua et al. 2025)
- **Implication**: The information encoding in PDFs
    is structurally inadequate for agent consumption.

### O4: PDF information gap is systematic
- **Statement**: Across 23 PaperBench papers (8,921
    rubric requirements), only 45.4% of reproduction
    requirements are fully specified in the PDF.
- **Evidence**: Own experiment -- info_gap_aggregate
- **Implication**: The PDF format is structurally
    incapable of serving as a self-contained
    reproduction specification.
... <!-- 4 observations total -->

## Gaps
### G2: Negative knowledge is systematically discarded
- **Statement**: Dead ends, rejected hypotheses, and
    convergence-critical tricks are lost to the
    narrative compression of the publication process.
- **Caused by**: O1
- **Why it matters**: Downstream agents waste compute
    re-exploring paths already proven fruitless.
... <!-- 2 gaps total -->

## Key Insight
- **Insight**: Separate research knowledge into four
    orthogonal layers -- structured scientific logic
    (Cognitive Layer /logic), minimal executable code
    (Physical Layer /src), preserved decision history
    (Exploration Graph /trace), and raw empirical
    results (Evidence Layer /evidence) -- to create
    a machine-executable knowledge package that
    eliminates both the Storytelling Tax and
    Engineering Tax.
- **Derived from**: O1, O2, O3, O4, G1, G2
```

A.5.4. COGNITIVE LAYER: LOGIC/SOLUTION/HEURISTICS.MD

Each heuristic records a design decision with its rationale, provenance (who introduced it), and sensitivity rating so that agents know which choices are safe to vary.

```
# Heuristics

## H04: Directional verification over exact matching
- **Rationale**: Legacy papers routinely omit details
    needed for exact reproduction. Verifying
    directional properties (A > B on metric X)
    demonstrates the code kernel captures the core
    algorithmic insight without requiring exact
    numerical matches.
- **Provenance**: user
- **Sensitivity**: medium
- **Code ref**: [paper/sections/protocol.tex]

## H12: Minimal kernel = algorithm notes with inline
       snippets, not raw code files
- **Rationale**: Full code dumps (200-700 lines)
    cause context dilution -- the agent spends
    tokens parsing boilerplate already described in
    official_solution_notes.md. Notes contain core
    algorithm with key code snippets inline,
    sufficient for comprehension while 5-10x smaller.
- **Provenance**: user-revised
- **Sensitivity**: high
- **Code ref**: [code/artifacts/rebench-*/src/
    kernel/official_solution_notes.md]
... <!-- 18 heuristics total -->
```

A.5.5. EXPLORATION GRAPH: TRACE/EXPLORATION_TREE.YAML

The exploration tree preserves decisions, dead ends, and experiments as a traversable graph. We show one node of each type from this paper's 94-node tree.

```
tree:
  - id: N04
    type: decision
    title: "Tripartite layer architecture"
    provenance: user
    timestamp: "2026-03-09"
    choice: >
      Three orthogonal layers: Cognitive (/logic),
      Physical (/src), Exploration Graph (/trace).
      Each addresses a distinct tax.
    alternatives:
      - "Two-layer (logic + code only)"
      - "Four-layer (separate evidence at top)"
      - "Flat structured document (single file)"
    evidence: >
      Three-layer separation achieves minimal
      representation while preserving all three
      dimensions that PDFs conflate.

  - id: N50
    type: dead_end
```

```
    title: "Trimming src/ alone does not recover
         Cat C from enrichment regression"
    provenance: ai-suggested
    timestamp: "2026-03-14"
    hypothesis: >
      Removing boilerplate code from src/ would
      reduce context dilution enough for Cat C
      (failure knowledge) to recover to ~80%.
    failure_mode: >
      Cat C stayed at 57.5% despite reducing src/
      from 56-104K to 20-36K per artifact. Remaining
      enrichment additions still dilute
      trace/exploration_tree.yaml content.
    lesson: >
      Context dilution for Cat C is more sensitive
      than expected. Even ~200 lines of structured
      markdown in src/ can push failure knowledge
      below the retrieval threshold.

  - id: N17
    type: experiment
    title: "24,008-run exploration waste analysis"
    provenance: ai-executed
    timestamp: "2026-03-12"
    result: >
      Analyzed 24,008 runs across 21 models,
      228 tasks. 59.2% of tokens wasted on
      dead-end exploration. 90.2% of cost goes
      to failed runs. Failed runs consume 113x
      more tokens than successful ones (median).
    evidence: [C13, C14, "code/eval/malt_analysis/
      exploration_tax_findings.json"]
... <!-- 94 nodes total -->
```

A.5.6. Exploration Graph: `trace/sessions/`

Each research session is logged as a structured YAML record capturing events, AI actions, files changed, claims affected, and open threads. Below is one session (abridged); the `session_index.yaml` file provides a one-line chronological summary of all 36 sessions.

```
# trace/sessions/2026-03-19_001.yaml
session:
  id: "2026-03-19_001"
  timestamp: "2026-03-19T02:00"
  summary: "BAM reproduction pilot: ARA 88.2% vs
    baseline 93.2% -- first paper where baseline leads"

events_logged:
  - type: experiment
    id: N65
    summary: "BAM ARA mega-task: 220.5/250 (88.2%
      weighted), 10/10 subtasks, 6.9h, 12.6M tokens"
  - type: experiment
    id: N66
    summary: "BAM baseline mega-task: 234/251 (93.2%
      weighted), 10/10 subtasks, 4.3h, 8.6M tokens"
  - type: observation
    id: O50
    summary: "ARA won T6 (non-Gaussian) 10/10 vs
      8.5/10 -- clearer hyperparameter specs"

ai_actions:
  - action: "Ran BAM ARA reproduction (10 subtasks)"
    files_changed:
      - "code/eval/reproduction/results/bam/"

claims_touched: [C05]

open_threads:
  - "Hyperparams buried in evidence/ not src/configs/
     -- move them and rerun (becomes N67)"
```

```
# trace/sessions/session_index.yaml (excerpt)
sessions:
  - id: "2026-03-12_004"
    summary: "Full 23-paper info gap analysis -- 8,921
      reqs, median 45.3% sufficient"
  - id: "2026-03-17_001"
    summary: "Pilot reproduction: ARA 74% vs baseline
      ~14%, 5x advantage on neural-score-estimation"
  - id: "2026-03-22_001"
    summary: "Self-expansion v2 (real code): 10/10
      subtasks -- v1 pseudocode got 1/10"
  - id: "2026-03-26_001"
    summary: "small_scaling_law extension v2: ARA 0.644
      vs baseline 0.806 -- baseline wins on loss
      calibration despite ARA finding near-optimal
      hyperparams"
... <!-- 36 sessions total -->
```

The complete artifact, including all 16 claims, 18 heuristics, 94 exploration nodes, and 36 session records, is available in the supplementary material at `ara/`.

B. ARA Compiler Details

The ARA Compiler (§3.2) is implemented as an *agent skill*: a self-contained, natural-language specification that, when loaded into any general-purpose coding agent's context, turns it into a domain-specialized compilation system. The skill prescribes *what* the agent should do and *what domain knowledge* it needs, but delegates all execution mechanics (model selection, tool dispatch, context management) to the host agent. This appendix reproduces the key elements of the skill specification; the complete definition is available in the supplementary code.

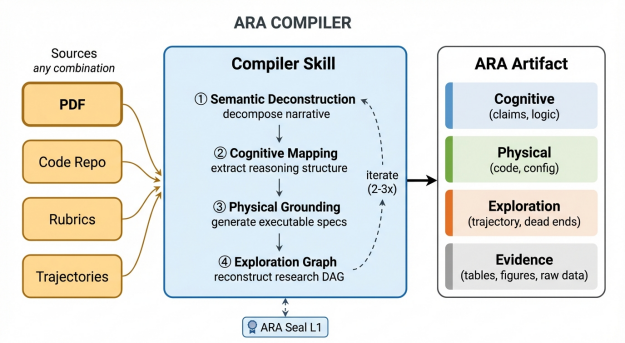


Figure 8. The ARA Compiler accepts any combination of research sources and guides a coding agent through four stages of top-down artifact compilation, iterating 2–3× with in-loop ARA Seal Level 1 validation until the output conforms to the protocol.

B.1. Design Rationale

Four invariants guide the Compiler's behaviour: what it must preserve, where it routes auxiliary sources, what it must reconstruct beyond plain extraction, and how quality is enforced.

High-fidelity preservation as a normative guarantee. A narrative paper compresses and selects; the Compiler decompresses and restores. Every numerical result, hyper-

parameter, architectural detail, and negative finding in the sources must appear *somewhere* in the artifact, and any PDF-accessible information missing from the ARA constitutes a compilation failure (evaluated in §4.1). Preservation is faithful to sources; *enrichment* (collective inference) separately surfaces cross-artifact patterns no single source expresses, distinguishing stated from inferred knowledge with provenance tags.

Auxiliary source routing. Beyond the PDF, research knowledge is distributed across complementary sources, each populating a distinct layer: GitHub repositories encode implementation decisions that prose omits and replace stubs with verified implementations in `/src`; expert-curated evaluation rubrics encode what domain practitioners consider the core claims and anchor `/logic` with claim decompositions; and recorded experimental trajectories preserve failure modes that papers systematically suppress and seed `/trace` with dead-end nodes the PDF omits. The Compiler routes each auxiliary source to the layer it most directly populates.

Knowledge lineage vs flat extraction. Plain-text extraction recovers content but not the cross-layer connections that lineage requires: parsing a PDF into Markdown populates four directories yet leaves them structurally isolated. The Compiler instead performs *forensic reconstruction* from sources where lineage exists only implicitly—scattered across prose, figure captions, appendix tables, and code comments—so an agent can trace any claim downstream to code or any number upstream to its hypothesis. Recovering this lineage, not populating layers, is the core compilation problem.

Quality enforcement: in-loop vs downstream. Quality is enforced in two stages. *During* compilation, the Compiler uses only ARA Seal Level 1 as an in-loop validation signal: schema conformance, cross-layer reference resolution, and required-field completeness, with failures returned as structured diagnostics that drive targeted fixes within the same agent conversation. *After* compilation, the finished artifact enters the downstream Seal pipeline (Appendix J), where Levels 2–3 evaluate its argumentative support and empirical reproducibility under the conditions a venue would impose.

B.2. Skill Specification

The Compiler skill specification (∼482 lines of natural language) is structured into five sections. When loaded into a host agent's context, it provides the full domain knowledge needed to produce a schema-conforming ARA. We reproduce representative elements below; the complete specification is available in the supplementary code.

Section 1: Workflow (lines 1–11). Defines the high-level pipeline: analyze → generate → validate → fix → iterate.

Section 2: Capability usage guidelines (lines 13–26). Specifies usage conventions for standard file operations: prefer `edit_file` over `write_file` for targeted fixes, prefer `write_file` for YAML files to avoid whitespace corruption. Instructs the agent to batch work: generate all files first, then validate once.

Section 3: ARA directory schema (lines 28–414). Defines the complete directory structure and field-level requirements for every file. This is the normative schema specification that the agent must follow. Key constraints include:

- **PAPER.md**: YAML frontmatter with title, authors, year, venue, DOI, domain, keywords, claims summary, and abstract. Body must include a Layer Index table listing every file with a one-line description.
- **claims.md**: Each claim requires Statement, Status, Falsification criteria, Proof (referencing experiment IDs, not file paths), Dependencies, and Tags.
- **experiments.md**: Declarative verification plans (setup, procedure, metrics, directional expected outcomes). Exact numerical results are *prohibited*—they belong exclusively in `/evidence/` to enable blind reproduction.
- **heuristics.md**: Each heuristic requires Rationale, Sensitivity (low/medium/high), Bounds, Code ref, and Source.
- **exploration_tree.yaml**: Nested YAML tree with typed nodes (`question`, `experiment`, `dead_end`, `decision`, `pivot`). Minimum 8 nodes with at least one `dead_end` and one `decision`. Dead ends must document hypothesis, failure mode, and lesson.
- **evidence/**: Every results table and quantitative figure must be reproduced with exact cell values—no rounding, no omission.

Section 4: 4-stage reasoning protocol (lines 416–454). The prompt mandates a structured thinking process before file generation:

```
# Your 4-Stage Reasoning Process

You MUST follow these 4 stages in order. Produce a
<thinking> block first with your reasoning for each
stage, then produce the files.

## Stage 1: Semantic Deconstruction
Strip the narrative "Storytelling Tax." Isolate:
- The core observations and gaps that motivate the work
- Mathematical formulations and equations
- Architectural specifications and component
    descriptions
- Experimental configurations (hyperparameters,
    hardware,
  datasets)
```

```
- Numerical results and benchmarks
- Citation dependencies and their roles
- Negative results and ablation findings

## Stage 2: Cognitive Mapping
Map deconstructed content to /logic:
- Extract motivation: observations (with numbers), gaps,
  the key insight, and assumptions
- Identify falsifiable claims (not opinions or vague
  statements)
- Define formal concepts with precise notation
- Populate solution/ (architecture, algorithm,
  constraints, heuristics)
- Construct typed dependency graph for related_work.md
- Ensure every claim has falsification criteria and
  proof pointers to experiment IDs
- Design declarative experiment plans: for each major
  claim, specify how an agent would verify it

## Stage 3: Physical Stubbing
Generate /src:
- Extract exact hyperparameter values into configs/
- Write code stubs with correct function signatures
  and types
- Specify environment (dependencies, hardware, seeds)
- Code should implement the NOVEL contribution,
  not boilerplate

## Stage 4: Exploration Graph Extraction
Reconstruct the research DAG as a nested YAML tree
for /trace:
- Identify the central research question(s) as root
  nodes
- Map experiments and their outcomes as child nodes
- Document dead ends from ablations and rejected
  alternatives as leaf nodes
- Record key design decisions with alternatives
  considered
```

Section 5: Output format and rules (lines 456–482). Specifies the XML-delimited output format for `batch_write_files`, lists all 15 mandatory files, and enforces nine invariant rules (e.g., "all numerical values must be EXACT as stated in the paper," "never hallucinate claims, results, or heuristics not in the paper").

C. ARA Live Research Manager Details

The Live Research Manager (§3.1) is the second agent skill in the ARA system. Like the Compiler, it is a self-contained natural-language specification that turns a general-purpose coding agent into a domain-specialized system; unlike the Compiler, it operates continuously alongside the researcher rather than as a one-shot compilation. This appendix expands on the design principles, cross-session mechanisms, and submission workflow summarized in §3.1.

Event taxonomy. The Event Router classifies each extracted event into one of seven types (Table 6) and tags it with provenance. Event payloads follow the protocol's factual-density requirement (§2.1): conversational prose is distilled into telegraphic, quantitative language before committing to the artifact. Trace events (decisions, experiments, dead ends, pivots) append to the Exploration Graph (`/trace/`); claims and heuristics enter the Cognitive Layer (`/logic/`); and events that resist classification are staged in `/staging/` for later promotion via the closure-driven mechanism described below.

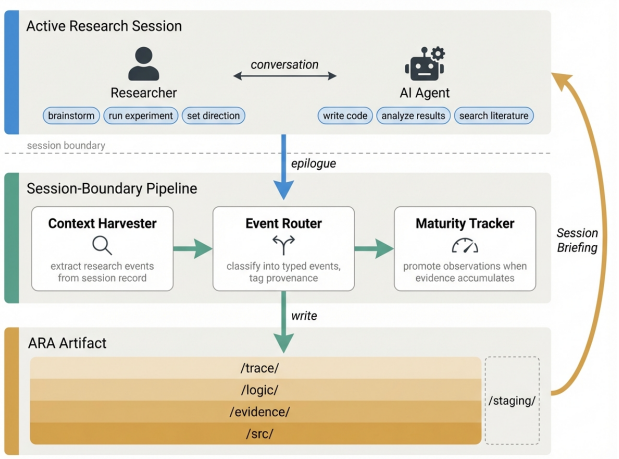


Figure 9. The Live Research Manager operates at session boundaries: a three-stage pipeline (Context Harvester → Event Router → Maturity Tracker) distills each researcher–agent conversation into typed events that accumulate across ARA layers over time.

C.1. Design Principle Rationale

The three design principles underlying the Live Research Manager (§3.1) are expanded below with full motivation.

P1. Silent, framework-independent integration. Documentation has traditionally been a *retrospective* activity: a context switch that introduces both friction and information loss. The system must integrate with any general-purpose coding agent (Claude Code, Cursor, Windsurf, or future frameworks) without custom SDKs, API bindings, or infrastructure changes. A natural-language specification that the agent reads into its context is the most portable interface: it requires nothing beyond the tool access agents already have, and artifact quality improves automatically as language models advance. The manager runs as a background process that silently collects research traces, constructing the artifact without interrupting active work or injecting prompts into an ongoing research conversation.

P2. Faithful epistemic provenance. AI-native research blurs the boundary between human insight and machine execution. The manager must objectively track *who did what*: distinguishing ideas explicitly stated by the researcher, suggestions inferred by the agent, actions the agent executed autonomously, and AI suggestions the researcher revised. Without such provenance, an artifact cannot faithfully represent the epistemic origin of its contents. A behavioral consequence:

Event Type	Structured Payload	Event Type	Structured Payload
`decision`	Choice, alternatives, evidence	`claim`	Statement, falsification criteria
`experiment`	Metrics, claim linkage	`heuristic`	Trick, sensitivity, bounds
`dead_end`	Hypothesis, failure mode, lesson	`observation`	Raw finding, awaiting classification
`pivot`	Trigger, rationale		

Table 6. Research event types and structured payloads emitted by the Event Router.

an `ai-suggested` event never auto-upgrades to a confirmed claim or decision until the researcher explicitly endorses it, preserving the epistemic integrity of the artifact even when the agent generates fluent but unconfirmed reasoning chains. The research process is inherently chaotic: a single session may interleave hypothesis formation, coding, debugging, and writing with no clear boundaries. The manager must translate this raw conversational stream into the structured ARA schema without losing information or imposing premature structure. Observations that are not yet classifiable should be staged rather than forced into categories, and knowledge should mature progressively, from hunches to typed events to formally bound claims, mirroring how research understanding actually develops.

P3. Comprehensive trajectory capture. Research is nonlinear and stochastic: hypotheses branch, experiments fail, directions are abandoned and revisited. The manager must capture this full trajectory, not just the successes that survive into a polished paper, but the dead ends, pivots, and intermediate observations that constitute the actual research process. Cross-layer bindings between claims, evidence, code, and decisions must be established at capture time while the conversational context is still available; post-hoc reconstruction from archived transcripts loses these causal chains.

C.2. Two-Phase Crystallization and Pivot Propagation

Artifact construction proceeds at two timescales (**P2**). *Continuously*, at every session boundary, the manager appends trace events to the Exploration Graph, recording how the researcher navigates the open-ended research landscape. *Periodically*, when a major milestone is reached—a hypothesis confirmed or refuted, a working prototype completed, a key design choice finalized—the Maturity Tracker *crystallizes* the accumulated observations into the more structured layers of the artifact: raw observations mature into formal claims in the Cognitive Layer (`/logic/`), working code is documented in the Physical Layer (`/src/`), and concepts are added to the knowledge index. This two-phase rhythm mirrors how research understanding actually develops: insights begin as scattered observations, and forcing premature structure would distort the record. When a pivot invalidates an earlier design choice, the manager propagates the change: it updates the affected artifact entries (claims, heuristics, configuration) to reflect the new direction while the Exploration Graph retains the original rationale and the reason for revision, so the project's intellectual history is preserved alongside its current state. At the start of each next session, the manager reads the artifact back and surfaces a structured briefing of open threads, recently crystallized claims, and unresolved contradictions, closing the loop with the researcher.

C.3. Closure-Driven Crystallization

Principle P2 requires that observations mature "as evidence accumulates," but *evidence accumulation* needs an operational definition. A counter-based threshold (promote after N references) is arbitrary; asking an LM in isolation "is this mature?" lacks grounding. We instead define maturity through *closure signals*: externally observable patterns in the researcher–agent conversation that indicate the researcher has treated an observation as settled. At each session boundary, the Maturity Tracker inspects the session record and promotes a staged observation when at least one closure signal is present.

Closure signal taxonomy.

Topic abandonment. The researcher has moved to a new topic without revisiting the observation in the subsequent k turns (default $k = 5$), and no pending question remains open on the original thread.

Verbal affirmation. The researcher explicitly endorses the observation ("yes, we'll go with X," "confirmed," or equivalent paraphrase), making the adoption decision first-person.

Empirical resolution. An experiment bound to the observation has produced a result and the researcher has commented on it; both supported and refuted outcomes are valid terminations (the refuted case promotes to a `dead_end`).

Artifact commitment. A downstream artifact now depends on the observation: code is merged, a config value is fixed, a subsequent claim uses it as a premise, or a design decision is documented as following from it.

Contradiction trigger. When a new observation contradicts one already staged or crystallized, the Maturity Tracker does not silently overwrite. Both entries are flagged, the contradiction is appended to the Exploration Graph as a `decision` node with unresolved status, and resolution is deferred to the next briefing where the researcher adjudicates explicitly.

C.4. Cross-Session Continuity

A stateless coding agent has no memory of previous conversations. If the manager simply reads the artifact at each session boundary, it knows *what* the artifact contains but not *why* it is organized the way it is—which classification choices were non-obvious, which observations were deliberately deferred, which patterns guided past merges and promotions. Without this self-awareness, the manager risks inconsistent classification, duplicate entries, and organizational drift across sessions.

We address this with two lightweight mechanisms. First, a *reasoning log* (`trace/pm_reasoning_log.yaml`) records the manager's own organizational decisions and their rationale at each session boundary—a compressed account of a few lines per session that gives the manager self-continuity without requiring access to raw conversation transcripts. Second, each session record includes a *key context* field: compressed summaries of the most important human–agent exchanges, preserving conversational nuance that would otherwise be lost when the raw conversation is no longer available. Together, these mechanisms ensure that the manager can read back not only the artifact's current state but also the reasoning chain that produced it, maintaining coherent organization across arbitrarily many sessions at negligible token cost.

D. Review System Details

The Review System is the third ARA skill alongside the Compiler (App. B) and the Live Research Manager (App. C). This appendix specifies the full design: the two design principles (§D.1), the four-level ARA Seal credential and its figure (§D.2), and the per-level checker implementation and failure taxonomy (§D.3). Empirical evaluation of the Seal (compiler convergence, mutation benchmark, execution reproducibility) is reported separately in App. J.

D.1. Design Principles

We derive two design principles below; cross-artifact review via agent-to-agent ARA comparison, which becomes meaningful only as the corpus reaches critical mass, is discussed in §M.

Figure 10. The ARA Seal is a four-level verification credential. Each level tests a progressively stronger property of the artifact, escalating in cost: structural integrity (seconds, deterministic), argumentative rigor (minutes, rubric-anchored agent), execution reproducibility (hours to days, sandboxed coding agent), and human judgment (days to weeks). Each level gates the next; passing the applicable levels issues a Seal Certificate that downstream agents check before investing compute.

P1. Automate the mechanical; reserve humans for judgment. Structural validity, internal consistency, and claim reproducibility are objective properties that either pass or fail, whereas significance, novelty, and taste require domain expertise. The review system should never ask a human to verify that Experiment E03 matches Claim C02 when a machine can do so in seconds; resolving all machine-checkable issues *before* the artifact reaches a human reviewer ensures that expert attention is spent exclusively on questions that genuinely require it.

P2. Reproducibility as a foundational requirement. "Code available upon request" nominally satisfies today's reproducibility bar; in an ARA-native system, reproducibility is a *machine-verified property* of the artifact itself. Passing ARA Seal Level 1 (structural integrity) is a submission requirement, and Level 2 (argumentative rigor) produces a structured critique before the venue spends compute on Level 3 (execution reproducibility), whose results are then attached to the Level 4 human review. Artifacts that fail structural checks, or whose claims remain obviously unsupported after Level 2 critique, do not advance.

D.2. The ARA Seal: Four-Level Verification Credential

A PDF paper earns trust indirectly: venue prestige, citation counts, and author reputation serve as proxies for quality, but none verify the work itself. Because an ARA encodes research as typed, machine-traversable data with explicit claim–evidence bindings, its quality becomes *directly verifiable*. The ARA Seal operationalizes this as a four-level verification protocol (Figure 10), where each level tests a progressively stronger property; per-level implementation and the failure taxonomy follow in §D.3.

Level 1 – Structural Integrity. Verifies that the artifact is well-formed and internally consistent: the directory ontology exists, all structured files conform to their schema (each claim carries `Statement`, `Status`, `Falsification criteria`, and `Proof`; each

heuristic carries `Rationale`, `Sensitivity`, and `Bounds`), and all cross-layer references resolve (experiment IDs in `claims.md` point to valid entries in `experiments.md`, code references trace to implementations in `/src`). The check is deterministic and runs in seconds.

Level 2 – Argumentative Rigor. Without executing any code or consulting external sources, a *Rigor Auditor* agent evaluates whether the content of a Level-1-valid artifact is epistemically sound along six objective dimensions, each scored on an anchored 1–5 rubric. The three load-bearing dimensions are *evidence relevance* (every claim's cited experiments substantively address what the claim asserts under type-aware entailment: causal claims require isolating ablations, generalization claims require heterogeneous test conditions, improvement claims require baseline comparisons), *falsifiability quality* (criteria are actionable, non-tautological, scope-matched, and independently testable without proprietary data), and *methodological rigor* (baseline adequacy, ablation coverage, statistical reporting, metric–claim alignment). Three further dimensions—scope calibration, argument coherence, and exploration integrity—are defined in App. J.2. Findings are tagged by severity (*critical*, *major*, *minor*, *suggestion*) with verbatim evidence spans and actionable suggestions; the overall grade is derived from the mean score and per-dimension floors. Every check reduces to a rubric-anchored property of the artifact's content, so Level 2 remains objective; significance, novelty, and taste are reserved for human reviewers at Level 4.

Level 3 – Execution Reproducibility. Verifies that the artifact's central claims reproduce empirically. The system selects claims by criticality (those in the contribution list, those anchoring the most downstream dependencies, or those flagged by the authors) and runs scaled-down directional checks (small data, few epochs, toy configurations) that test whether claimed properties hold qualitatively rather than reproducing exact numbers. The verifying agent is isolated from the artifact's evidence layer: it receives only the code kernel and algorithm descriptions, never the reported numbers, preventing fabrication by copying expected outcomes. Venues set a compute budget; claims that exceed it are flagged as *unverified*. Full-scale reproduction (original datasets, full training runs, exact metric recovery) is optional and typically post-acceptance or community-driven; results are appended to the living Seal Certificate. Beyond reproduction, the agent assesses experimental comprehensiveness: whether ablations are present for each design choice the claims attribute non-trivial impact to; whether experimental conditions cover the claimed generality or are cherry-picked from favorable settings; and whether there are undocumented heuristics in the code that do not appear in the artifact's cognitive layer.

Level 4 – Human Judgment. The only level that cannot be automated: reviewers receive the submission alongside the CI Report (Levels 1–2) and Empirical Review Report (Level 3), and their role shifts from verification to judgment. They focus on questions no machine can answer: is this contribution significant; is the core insight genuinely novel or an incremental recombination; is this the right formulation of the problem; what are the ethical implications and potential for misuse. Where AI reviewer findings are contested by the authors, humans adjudicate with the full audit trail (CI Report, rigor report, Empirical Review Report, and per-finding evidence spans) available for inspection. Human reviewers write their reviews in the same typed format used by the automated levels (findings linked to specific ARA components: claim IDs, experiment IDs, exploration-tree nodes), so all feedback is actionable and traceable, and downstream agents can read the review history of any artifact as structured data rather than free prose.

Passing the applicable levels issues a **Seal Certificate**: a signed record of artifact ID, verification level achieved, timestamp, environment hash, and per-claim reproduction outcomes. Downstream agents check the certificate before investing compute, avoiding redundant re-verification.

D.3. Seal Implementation Details

Per-level checkers. Each verification level is implemented as an automated checker:

- **Level 1 (Structural Integrity)**: A Python script verifies (a) the existence of mandatory directories (`/logic`, `/src`, `/trace`, `/evidence`), (b) the presence of all mandatory files including `PAPER.md` with valid YAML frontmatter, `problem.md`, `claims.md`, `experiments.md`, and all `solution/` files, (c) schema conformance of every structured file (e.g., each claim must have `Statement`, `Status`, `Falsification criteria`, and `Proof`; each experiment must have `Verifies`, `Setup`, `Procedure`, and `Expected outcome`; each heuristic must have `Rationale`, `Sensitivity`, and `Bounds`), (d) minimum counts ($\geq$ 5 concepts, $\geq$ 3 experiments, $\geq$ 8 exploration tree nodes with at least one `[dead_end]` and one `[decision]`), and (e) cross-layer reference resolution: every experiment ID referenced in `claims.md` `Proof` fields resolves to an entry in `experiments.md`; every claim ID referenced in `experiments.md` `Verifies` fields resolves to an entry in `claims.md`; every `code_ref` in `heuristics.md` points to a valid module in

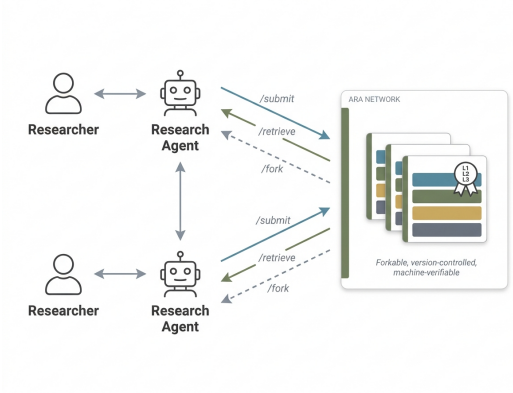


Figure 11. The (Human+AI)^2 research network. Each researcher works through a research agent that interfaces with a shared ARA network via `/submit`, `/retrieve`, and `/fork`; agents may also collaborate directly.

`/src`; components declared in `architecture.md` have corresponding code stubs; claim references in `exploration_tree.yaml` resolve to valid claim IDs.

- **Level 2 (Argumentative Rigor)**: Without executing code or consulting external sources, the Rigor Auditor evaluates the artifact's content on six objective rubric-anchored dimensions (evidence relevance, falsifiability quality, scope calibration, argument coherence, exploration integrity, methodological rigor), each scored on a 1 to 5 scale. The output is a rigor report keyed to specific ARA components, with severity-ranked findings, verbatim evidence spans, and an overall grade derived from the mean score and per-dimension floors.

- **Level 3 (Execution Reproducibility)**: A coding agent reads the ARA and attempts to reproduce claims using the code kernel. LLM-generated test cases verify directional properties of the paper's claims. This is the same protocol used in the reproduction evaluation (§4.2).

- **Level 4 (Human Judgment)**: No automated checker; human reviewers receive the CI Report and Empirical Review Report and submit structured reviews keyed to specific ARA components. The Seal Certificate records the assigned review decision alongside the automated level results.

Failure taxonomy. For each Seal level failure, we record the specific check that failed and classify it into one of the following categories: *missing file*, *missing field*, *dangling reference*, *type mismatch*, *dependency resolution failure*, *execution error*, or *nondeterminism*.

E. The Human–AI Interface

The three ecosystem skills described in §3 compose into a scientific communication system whose primary object is no longer the static document but the ARA itself: a single canonical artifact that humans on each end direct agents to author, certify, render, and extend.

Producer side. Researchers no longer work toward papers; they pursue questions, and the paper-as-output accrues automatically as an ARA along the way. The Live Research Manager folds each decision, dead end, and confirmed claim into the artifact during ordinary work; the Compiler imports legacy sources on demand; and at any milestone the artifact is routed through the Seal pipeline and registered publicly. At that point, another team can fork a passing artifact, extend a claim, retain attribution to the parent, and submit the diff for re-review—making publishing a Git-like operation in which every contribution is an executable diff.

Consumer side. Because the ARA is canonical, an agent renders it on demand into whatever surface the reader needs (paper, video, slides, interactive demo, or grounded dialogue), shaped by the reader's expertise, attention budget, and intent. Reviewers consume Seal-attested artifacts through their preferred surfaces, and downstream agents read ARAs as structured baselines, training environments, or starting points for new questions.

Compounding contributions. With both ends agent-mediated and the artifact as the only persistent state, contributions compound at the level of artifacts rather than sentences. The result is a queryable scientific commons in which the cost of understanding, reproduction, and extension falls with each new artifact admitted rather than rising against it.

F. Evaluation Setup

This appendix defines the shared evaluation infrastructure used by the four evaluation appendices that follow: the test corpus (App. F.1) consumed by the Understanding (App. G), Reproduction (App. H), and Review System (App. J) experiments, and the shared agent and judge configurations (App. F.2). Each subsequent evaluation appendix specifies only its own additional setup (task design, scoring formula, harness). The RE-Bench tasks used by the Extension experiment (App. I) are documented separately in App. I.1.

F.1. Test Corpus

Selection criteria. We draw our evaluation corpus from PaperBench (Starace et al., 2025), which provides expert-authored hierarchical reproduction rubrics for ICML 2024 papers. We adopt all 23 papers in PaperBench's public

release as our PaperBench corpus, with no exclusions: every paper satisfies our three required properties: (1) peer-reviewed at a top ML venue (ICML 2024 Spotlight/Oral, with two NeurIPS 2024 workshop development papers) with a publicly available PDF; (2) spanning diverse ML subfields to test breadth across the discipline; (3) accompanied by a PaperBench rubric with fine-grained leaf requirements that enables quantitative evaluation of reproduction fidelity. We supplement this corpus with 7 open-ended R&D tasks from RE-Bench (Wijk et al., 2025), yielding 30 evaluation targets with 450 questions total. Table 7 summarises the two benchmarks side by side.

PaperBench (Starace et al., 2025)	**RE-Bench** (Wijk et al., 2025)
Papers: 23 peer-reviewed ML papers across diverse subfields	**Tasks:** 7 R&D hill-climbing tasks on well-defined objectives
Scale: 8,921 expert-authored rubric requirements	**Scale:** 24,008 agent runs; 46,303 failure episodes
Contents: PDF + companion GitHub repo (15/23 papers); expert rubrics encode hyperparameter values, implementation tricks, and configurations absent from the paper; failure knowledge rarely preserved in published papers	**Contents:** Starter codebase per task; METR MALT transcripts with full real agent successful and failure trajectories retained per run
Scoring: Expert hierarchical rubric (yes / partial / no)	**Scoring:** Automated continuous objective; no human judgment
Used in: Understanding (§4.1, Cat. A & B); Reproduction (§4.2)	**Used in:** Understanding (§4.1, Cat. C); Extension (§4.3)

Table 7. Benchmark characteristics. PaperBench supplies configuration depth via expert rubrics; RE-Bench supplies trajectory depth via MALT failure traces. These are the two source-side enrichments ARA is built to exploit.

Paper list. Table 8 lists the 23 PaperBench papers used across the understanding and reproduction evaluations. Note that PaperBench's own protocol blacklists author code; we relax this for the baseline so it has the strongest possible footing (PDF + companion GitHub), making "repo availability" a property of our harness rather than of PaperBench. Of the 23 papers, 15 are included in the reproduction experiment; the remaining 8 are excluded because faithful end-to-end reproduction exceeds our per-task compute budget or requires specialized infrastructure outside our evaluation harness (e.g., multi-day CLIP adversarial fine-tuning, Isaac Gym RL, full ImageNet coreset sweeps, large multi-model benchmark suites). These 8 papers participate only in the understanding evaluation.

Short Name	**Domain**	**Repro**
adaptive-pruning	Efficiency	Yes
all-in-one	Multi-task	Yes
bam	LLM alignment	Yes
bbox	Black-box LLM	Yes
bridging-data-gaps	Data	No
fre	RL	Yes
ftrl	Online learning	Yes
lbcs	Calibration	No
lca-on-the-line	Prediction	No
mechanistic-understanding	Interpretability	Yes
pinn	Scientific ML	Yes
rice	Retrieval	Yes
robust-clip	Robustness	No
sample-specific-masks	Data augmentation	Yes
sapg	Optimization	No
self-composing-policies	Continual RL	Yes
self-expansion	Self-training	Yes
semantic-self-consistency	Evaluation	No
seq.-neural-score-est.	Score estimation	Yes
stay-on-topic-w/-CFG	Generation	No
stochastic-interpolants	Generative	Yes
test-time-model-adapt.	Adaptation	Yes
what-will-my-model-forget	Continual learning	No
Total: 23		15

Table 8. Test corpus: 23 PaperBench papers from ICML 2024 spanning diverse ML subfields. The "Repro" column indicates inclusion in the reproduction experiment (requires companion code). All 23 papers participate in the understanding evaluation.

Corpus diversity. The 23 papers span diverse ML subfields: efficiency, alignment, interpretability, RL, scientific ML, generative models, optimization, retrieval, evaluation, and adaptation. While all papers come from a single venue (ICML 2024), they vary substantially in methodological complexity—from systems-oriented efficiency papers with minimal formal analysis to theory-heavy contributions (PINN, stochastic interpolants)—heuristic density, paper length, and contribution type (new architectures, training recipes, algorithms, analysis frameworks). The corpus includes papers that are deliberately challenging for structured extraction: those whose contributions are analysis rather than methods (mechanistic-understanding), multi-component pipelines (all-in-one), and papers with complex combinatorial experiment matrices (PINN with 1,273 leaf requirements).

Asset licenses. Table 9 lists every external asset used in this work and its license. Each is used within the bounds of its stated terms; PaperBench, RE-Bench, and the companion repositories of the 23 ICML 2024 papers are accessed only as already-public artifacts. Anthropic API usage complies with the Anthropic Terms of Service.

Aggregate compute budget. Compute is dominated by LLM API calls; non-LLM compute is consumer-grade.

Asset	Source	License
PaperBench (papers + rubrics)	Starace et al. (2025)	MIT (code) / CC-BY-4.0 (rubrics)
RE-Bench task suite	Wijk et al. (2025)	MIT
METR `eval-analysis-public` (MALT traces)	METR	MIT
23 ICML 2024 source papers	PaperBench corpus	Per-paper (predom. CC-BY-4.0)
15 companion GitHub repos	PaperBench corpus	Per-repo (MIT, Apache-2.0, BSD-3)
Claude Agent SDK	Anthropic (2025b)	MIT
Anthropic Claude API (Sonnet 4.5/4.6, Opus 4.6)	Anthropic	Anthropic Terms of Service
Newly released by this work:		
ARA protocol & schema	this paper	CC0
ARA Compiler / LRM / Seal Level 1–3 (code)	this paper	CC0
30 compiled ARAs, 450-question bank, 150-subtask harness, RE-Bench extension harness	this paper	CC0

Table 9. External assets used and new assets released. Per-paper and per-repo licenses for the PaperBench corpus are enumerated in the supplementary material.

Token totals are estimated from observed per-call usage logs and rounded to the nearest 10 M; USD figures use Anthropic's posted rates at the time of submission and are rounded to the nearest $100. Table 10 reports the per-experiment breakdown; the total project cost is approximately **9,000 USD** of API spend and **200 wall-clock hours** on a single 32-core x86 host with 128 GB RAM, one NVIDIA L40S GPU (48 GB), plus a SLURM partition with single-GPU NVIDIA H100 (80 GB) nodes used only for the RE-Bench extension experiment. Exploratory and failed runs that did not appear in the paper add roughly $1.5\times$ on top of the reported figures (chiefly compilation pipeline iterations and prompt tuning for the Compiler).

F.2. Shared Agent and Judge Configuration

The Understanding (App. G), Reproduction (App. H), and Review System (App. J) experiments share an evaluation agent and judge configuration; the Extension experiment (App. I) uses a custom SLURM-launched harness documented in App. I.4.

Evaluation agent. For each (target, format, query) triple, we instantiate a fresh sub-agent (Claude Sonnet 4.6). The agent receives the format under test (the full ARA directory for the ARA condition, or the PDF plus companion GitHub repository for the baseline) and a single query. Each query is answered independently with a fresh context to prevent information leakage across queries.

Judge model and grading scale. Outputs are scored on a ternary scale by an independent judge (Claude Opus 4.6) against a gold reference:

- **Correct (1.0)**: The answer matches the ground truth in substance. Minor phrasing differences are acceptable; numerical values must be exact.
- **Partial (0.5)**: The answer conveys the main insight but misses key sub-details.
- **Incorrect (0.0)**: The answer contains a factual error, contradicts the gold answer, or hallucinates an answer to an unanswerable question.

Each evaluation appendix specifies how this scale composes into its experiment-specific aggregate metric (e.g., per-category accuracy in App. G, difficulty-weighted success rate in App. H).

G. Understanding Evaluation

This section reports the methodology and per-stratum results for the Understanding experiment (§4.1). The shared test corpus is in App. F.1.

G.1. Question Bank and Grading Rubric

Question generation. For each of the 30 evaluation targets (23 PaperBench papers, 7 RE-Bench tasks), we generate 15 questions across three categories: 10 Category A (fidelity: information preservation), 5 Category B (configuration and detail recovery) for PaperBench papers, or 5 Category C (failure and exploration knowledge) for RE-Bench tasks. Questions are designed to require specific, unambiguous answers or verifiable outputs. We avoid opinion questions ("Is this architecture good?") and questions requiring external knowledge not in the paper ("How does this compare to BERT?").

Category A question templates (10 per paper).

- **Architecture & Method** (3 questions): "What is the [specific structural detail]?" (e.g., "How many layers does the encoder have?", "What are the inputs and outputs of

Experiment	Tokens	Wall	API $	Hardware
Understanding (§4.1)	∼110 M	30 h	∼1,500	x86 host (no GPU)
900 sub-agent runs (Sonnet 4.6)	100 M			
900 judge calls (Opus 4.6)	9 M			
Reproduction (§4.2)	∼540 M	60 h	∼5,000	x86 host + 1×L40S (48 GB)
30 paper-runs, 14–20 M tok each	510 M			
1,743 rubric judge calls (Opus 4.6)	30 M			
Extension (§4.3)	∼40 M	80 SLURM-h	≤500	1×H100 (80 GB) per task
5 tasks × 2 formats, Sonnet 4.6, 8 h cap, $50/run cap				
Extension sensitivity (Sonnet 4.5)	∼15 M	32 SLURM-h	≤200	1×H100 (80 GB) per task
2 tasks × 2 formats				
Compilation pipeline (30 ARAs)	∼200 M	60 h	∼2,000	x86 host (no GPU)
Compiler + per-MALT-run extraction sub-agents (Sonnet 4.6)				
Reported total	∼900 M	∼260 h	∼9,200	
Including exploratory/failed runs (1.5×)	∼1.4 B	∼390 h	∼13,800	

Table 10. Aggregate compute and API spend across all experiments. Local GPU usage is light: the L40S is used only for the few PaperBench reproductions that require local inference (e.g., `fre` 17-model PyTorch reimplementation at 1.8 GB peak GPU memory); H100 nodes for RE-Bench tasks rarely exceed 20 GB GPU memory (Triton kernels and small-scale GPT training).

the multi-head attention module?")

- **Hyperparameters & Configuration** (2 questions): "What [training/optimization detail] is used?" (e.g., "What optimizer is used and what are its hyperparameters?", "What is the batch size in tokens?")
- **Results & Claims** (3 questions): "What [metric] does the [model variant] achieve on [benchmark]?" (e.g., "What BLEU score does the base model achieve on WMT 2014 EN-DE?")
- **Rationale & Design Decisions** (2 questions): "Why was [design choice] made instead of [alternative]?" (e.g., "Why is scaled dot-product attention used instead of additive attention?")

Category B question templates (5 per PaperBench paper).

- **Implementation** (2 questions): "Implement [specific module] with the correct [dimensions/activations/structure]."
- **Configuration Recovery** (2 questions): "Write the [optimizer/training/data] configuration with the exact parameters specified."
- **Debugging & Troubleshooting** (1 question): "[Failure scenario]—identify the cause and fix it."

Category C question templates (5 per RE-Bench task).

- **Dead-End Knowledge** (3 questions): "What approaches have been tried and failed?" / "What is the documented failure mode of [approach]?"
- **Exploration History** (2 questions): "What alternatives were considered for [decision]?" / "What lesson was learned from the [dead-end] attempt?"

The shared evaluation agent (Sonnet 4.6) and judge (Opus 4.6) configuration is in App. F.2.

G.2. Per-Category Result Analysis

This subsection unpacks the three per-category results summarized in §4.1 and Table 1 into the specific structural mechanisms that produce each gain.

Category A: fidelity at lower cost via progressive disclosure. ARA preserves PDF-recoverable information with high fidelity while requiring fewer tokens to retrieve. On PaperBench, ARA achieves 96.7% vs. 89.8% for the baseline while consuming 12% fewer tokens per question (86.3K vs. 97.7K). The structural explanation is progressive disclosure: ARA's PAPER.md provides a layer index that directs the agent to the relevant file (e.g., `evidence/tables/` for numerical results, `logic/solution/algorithm.md` for method details), whereas the PDF agent must scan the entire document for each query. On RE-Bench, where the baseline reads only the synthesized polished paper rather than a real publication, ARA's accuracy advantage widens to 92.1% vs. 51.4%: the synthesized writeup omits much of the technical detail that the artifact's structured layers preserve. The headline finding is that structured organization improves accuracy while keeping token usage comparable, because ARA's layer taxonomy turns linear search into indexed lookup.

Category B: configuration recovery via centralized configs. The rubric-aligned questions probe fine-grained experimental details (hyperparameter values, environment specifications, preprocessing steps) that PaperBench rubrics demand but papers systematically omit (26.2% of all gaps are missing hyperparameters; see Appendix A.2). The baseline's 67.8% reflects successful code-repository mining: given a dedicated sub-agent per question, it can grep through the companion GitHub repo for many configuration values. ARA's `src/configs/` and `logic/requirements.md` layers, however, centralize this knowledge in human-readable files, raising accuracy to 92.6% at comparable token usage (183K vs. 178K tokens per question): the agent reads a structured config file rather than searching a scattered codebase. The remaining gap to 100% reflects details genuinely absent from both the paper and its repository, which the Compiler cannot synthesize.

Category C: failure knowledge has no analogue in the baseline. ARA reaches 81.4% on failure-knowledge questions while the baseline manages only 15.7%; the synthesized polished papers contain almost no record of failed approaches, dead-end configurations, or intermediate results that the trace layer preserves. The baseline's low token usage per question (58.0K) reflects this poverty: agents quickly determine the information is absent and return short answers, spending minimal tokens on fruitless search. ARA agents consume more tokens per question (139.3K) but productively explore the exploration tree to find answers. This category provides the clearest evidence for preserving negative knowledge: information that narrative formats systematically discard accounts for the largest single accuracy gap in the entire evaluation.

G.3. Statistical Details

A McNemar test on the 450 paired outcomes yields $\chi^2 = 95.15$, $p < 10^{-10}$ overall: ARA answers 141 questions correctly that the baseline misses, while the baseline answers only 18 that ARA misses. By category, the ARA advantage is highly significant for all three categories: Category A (+14.8%), Category B (+24.8%), and Category C (+65.7%, dominated by the absence of exploration knowledge in baseline sources).

Difficulty stratification. Stratified by question difficulty, ARA leads across all tiers: T1 (explicit) questions (ARA 97.3%, BL 83.8%; $n = 74$), T2 (scattered) questions (ARA 95.6%, BL 79.0%; $n = 193$), and T3 (implicit) questions (ARA 91.0%, BL 60.5%; $n = 172$). On unanswerable questions ($n = 26$), ARA achieves 92.3% abstention accuracy vs. 86.5% for the baseline. The difficulty gradient is expected: T2 and T3 questions require assembling scattered information or reasoning about implicit assumptions, where structured representations provide the greatest advantage.

Token usage–difficulty interaction. The per-question token data reveals that ARA's progressive disclosure architecture creates an adaptive search pattern: ARA agents consume 60.9K tokens/Q on T1 (explicit) questions, 95.5K on T2 (scattered), and 152.7K on T3 (implicit), adapting search depth to question complexity. Baseline agents, by contrast, show a flatter profile across difficulty tiers (82.8K–118.2K tokens/Q), because linear PDF scanning does not benefit from question-aware navigation. ARA consumes fewer tokens than the baseline on T1 (27% less) and T2 (13% less), and invests more on T3, while being substantially more accurate at every tier.

Benchmark group breakdown. On PaperBench papers ($n = 345$), ARA achieves 95.4% vs. 82.5% at comparable token usage. On RE-Bench tasks ($n = 105$), the accuracy gap widens (ARA 88.6% vs. BL 39.5%), driven by Category C questions where the baseline has no access to failure knowledge.

H. Reproduction Evaluation

This section reports the task design, scoring, and per-paper analysis for the Reproduction experiment (§4.2).

H.1. Reproduction Task Design and Scoring

Task curation. Each of the 150 reproduction tasks specifies a single model, a single method, and 5–15 rubric leaf requirements as success criteria, with difficulty stratified per paper (≥3 easy, ≥3 medium, ≥3 hard; aggregate: 50 easy, 49 medium, 51 hard). Tasks describe *what* to reproduce, not how—the agent decides its own implementation strategy. Within each paper, the 10 subtasks form a single *mega-task*: the agent receives all subtasks ordered by difficulty (easy → medium → hard) and builds cumulatively, naturally reusing prior work as a human researcher would.

Scoring formula. The primary metric is the **difficulty-weighted success rate**: $\sum_i s_i \cdot w_{d_i} \;/\; \sum_i m_i \cdot w_{d_i}$, with $w_d \in \{1, 2, 3\}$ for easy, medium, and hard subtasks, where s_i is the subtask score (sum of requirement weights for *yes* $+ 0.5\times$*partial*) and m_i the maximum possible. Easy subtasks (setup, model instantiation) are necessary but not discriminative; most agents complete them regardless of source material, while harder subtasks (training, ablation, cross-method comparison) are where structured information provides the most leverage. We also report the flat (unweighted) rate and per-difficulty breakdowns.

Statistical significance. A Wilcoxon signed-rank test on the 15 paired per-paper weighted scores yields $p = 0.028$:

ARA wins on 8 papers, ties on 5, and the baseline leads on 2. The sign pattern (8–2) is itself statistically improbable under the null hypothesis of no difference ($p = 0.039$, exact binomial), confirming that the aggregate advantage is not driven by a single outlier paper.

H.2. Per-Paper Reproduction Analysis

This section provides the detailed per-paper analysis for the reproduction experiment (§4.2).

Per-difficulty analysis. The aggregate per-difficulty pattern (ARA 85.1% vs. baseline 80.2% on easy, 68.5% vs. 62.9% on medium, 54.5% vs. 46.0% on hard) is visualized in main-text Figure 5. Figure 12 resolves this aggregate to the per-paper level, and Table 11 provides the full per-paper, per-difficulty success rates underlying both.

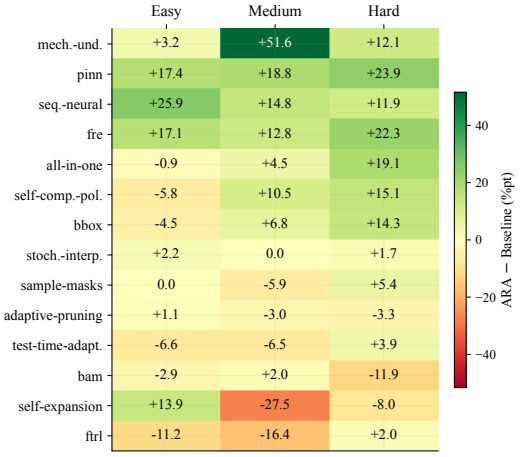


Figure 12. Per-paper ARA − baseline delta (percentage points) on each difficulty stratum, sorted by mean advantage. Green indicates ARA wins, red indicates baseline wins. Gains concentrate in the medium and hard columns across most papers; the few baseline wins are confined to a small set, most prominently `self-expansion` and `ftrl`.

Large ARA wins. The papers with the largest ARA advantages—`fre` (+21.3%), `mechanistic-understanding` (+20.7%), and `pinn` (+19.5%)—share complex multi-step training pipelines with non-obvious hyperparameter interactions that PDFs describe only at a high level. The `fre` ARA agent reimplemented the original JAX codebase in PyTorch (1.8 GB GPU vs. JAX's 30.8 GB), trained 17 models across three domains, and completed all medium and hard subtasks; the baseline agent struggled with the JAX environment and completed only 3 training attempts before exhausting its budget. The five newly added papers reinforce these patterns: `all-in-one` (+16.0%) and `ftrl` (+6.1%) show clear ARA advantages on hard tasks, while `stochastic-interpolants` (+0.5%) and `test-time-model-adaptation` (+0.3%) are ties with comparable performance from both sources.

Baseline wins and ties. The one clear baseline win is `self-expansion` (−7.3%), where the ARA agent exhibited result fabrication—reporting identical accuracy values across all configurations—detected by the blinded judge. Among the narrow ties, `adaptive-pruning` (−2.3%) and `rice` (−1.9%) both have strong companion repositories with runnable training scripts; the baseline's code access partially compensates for the PDF's information gaps. On `rice`, ARA achieves comparable quality with 2.5× less compute (3.7h vs. 9.1h, 131K vs. 195K tokens), suggesting efficiency gains even when final scores are similar.

Result fabrication. Two baseline runs (`bbox`, `mechanistic-understanding`) exhibited result fabrication—reporting plausible but uncomputed values when unable to complete training—detected by the blinded judge. Across all 15 papers, fabrication occurred in 2 baseline runs and 1 ARA run (`self-expansion`), suggesting that structured artifacts generally provide sufficient grounding to prevent hallucinated results, though they are not immune.

I. Extension Evaluation

This appendix documents the methodology behind §4.3: which RE-Bench tasks we use and why (App. I.1), how each ARA is compiled from official solutions and prior MALT trajectories (App. I.2), how the polished paper-arm baseline `paper.md` is generated (App. I.3), the engineering of the agent harness (App. I.4), how we extract canonical scores from agent traces (App. I.5), and the per-task case studies and trace evidence (App. I.6).

I.1. Task Selection

Of the 7 RE-Bench (Wijk et al., 2025) tasks, we use 5 in the extension evaluation. Table 12 lists the score formula, direction, on-task starting baseline, RE-Bench reference score, hardware requirement, and the model coverage of each task's METR MALT corpus.

The bottom two tasks are excluded because their MALT corpora cannot supply a usable failure-trace layer for the experiment. `optimize_llm_foundry` has no published MALT corpus at all, so `trace/` would be empty by construction. `small_scaling_law`'s MALT corpus does exist but is structurally inadequate: it predates Claude-4 (only Claude-3.5/3.7 Sonnet and OpenAI models), it is sparse, and the runs that do exist are dominated by trivial parameter-grid sweeps with no recorded strategic exploration or named dead ends. An extraction pipeline run on those runs produces effectively empty

	ARA (%)			Baseline (%)			Weighted	
Paper	Easy	Med.	Hard	Easy	Med.	Hard	ARA	Base
adaptive-pruning	90.9	80.0	31.7	89.8	83.0	35.0	63.5	65.8
all-in-one	90.4	92.0	61.1	91.3	87.5	42.0	72.8	60.4
bam	97.1	97.1	77.6	100.0	95.1	89.5	88.2	93.2
bbox	93.3	59.5	31.6	97.8	52.7	17.3	49.8	40.8
fre	79.3	45.4	50.9	62.2	32.6	28.6	53.2	34.4
ftrl	25.0	38.0	32.0	36.2	54.4	30.0	33.0	38.8
mechanistic-und.	85.7	88.3	67.1	82.5	36.7	55.0	76.2	55.0
pinn	96.2	93.8	89.8	78.8	75.0	65.9	92.2	71.0
rice	72.3	72.1	65.8	74.1	73.9	68.0	69.9	71.8
sample-specific-masks	95.6	29.8	31.1	95.6	35.7	25.7	42.7	42.3
self-composing-pol.	87.5	46.5	52.3	93.3	36.0	37.2	57.3	47.8
self-expansion	34.7	37.5	39.3	20.8	65.0	47.3	38.2	48.9
sequential-neural	95.2	68.3	51.6	69.3	53.5	39.7	64.8	49.3
stochastic-interpolants	97.7	100.0	74.1	95.5	100.0	72.4	86.7	85.3
test-time-model-adapt.	87.8	43.5	19.8	94.4	50.0	15.9	35.1	35.0
Mean	**85.1**	**68.5**	**54.5**	**80.2**	**62.9**	**46.0**	**64.4**	**57.4**

Table 11. Per-paper reproduction success rates (%) by difficulty level. Easy, medium, and hard columns show the unweighted success rate within each difficulty tier; the final two columns show the difficulty-weighted rate ($1:2:3$ weighting). Rice per-difficulty values are interpolated from the weighted score and overall rate, as its per-difficulty JSON entry was recorded separately.

Task	**Score formula**	**Dir.**	**Start**	**Ref.**	**Hardware**	**Claude-4 MALT**	**Used**
`triton_cumsum`	$\log(t_{\text{ms}})$	↓	1.56	0.47	1×H100	22 (O13/S9)	yes
`restricted_mlm`	$\log(\ell - 1.5)$	↓	1.81	1.13	2×H100 80 GB	22 (O11/S11)	yes
`fix_embedding`	$\log(\ell_{\text{val}} - 1.5)$	↓	2.20	0.26	1×GPU	19 (O10/S9)	yes
`nanogpt_chat_rl`	avg. pairwise win-rate	↑	0.54	0.85	1×GPU + judge	18 (O12/S6)	yes
`rust_codecontests`	$n_{\text{solved}}/165$	↑	0.00	0.13	CPU + LLM API	12 (O6/S6) + 10[†]	yes
`small_scaling_law`	$1-(\epsilon_\ell+\epsilon_p)$	↑	0.24	0.84	1×GPU	0[‡]	no
`optimize_llm_foundry`	$\log(t_s)$	↓	5.60	4.54	4×H100	0[§]	no

Table 12. RE-Bench task card with extension-evaluation status. *Score formula* is transcribed verbatim from `metr-re-bench/ai_rd_<task>/ai_rd_<task>.py`; ℓ denotes validation loss, t wall-clock time, n_{solved} the count of correctly solved problems, ϵ_ℓ/ϵ_p loss/parameter prediction errors. *Dir.*: score orientation. *Start*: score of the unmodified starter codebase. *Ref.*: best score reported in the original RE-Bench evaluation. *Claude-4 MALT*: count of Claude-4 (Opus + Sonnet) runs in the METR MALT corpus, broken down as O⟨Opus⟩/S⟨Sonnet⟩. [†] `rust_codecontests` also has a 10-run `claude-3-7-sonnet` supplement that uses the same scoring scaffold. [‡] only Claude-3.5/3.7 and OpenAI runs. [§] no MALT corpus published.

`trace/` and `evidence/` layers and neuters the experimental contrast against the paper-only baseline. We defer both tasks to follow-up work that re-runs MALT collection on Claude-4 agents (`optimize_llm_foundry`) or on tasks whose strategy space elicits substantive recorded exploration (`small_scaling_law`).

I.2. ARA Construction Pipeline

Each RE-Bench ARA is compiled from two sources: the official reference solution (copied verbatim into `src/`) and the task's METR MALT transcripts (extracted under a beat-reference filter into `trace/` and `evidence/`). The compiler is task-agnostic with per-task knobs (score formula and direction, MALT JSONL path, dev-history schema, known hazards) collected in per-task cards; the orchestrator procedure and shared sub-agent prompt live in `code/rebench-pipeline/`.

Pipeline. The orchestrator first lifts the official solution into `src/` and the reference-derived knowledge (mathematical formulation, algorithm, heuristics, baseline tables, dev-history nodes) into `logic/` and `evidence/`, with each node tagged `source: official-solution`. It then fans out one extraction sub-agent per MALT run; each sub-agent reads its run in full (no truncation, no chunk skipping) and emits a bundle of trace nodes, evidence rows, and insights. As sub-agents complete, the orchestrator merges their outputs into the artifact: deduplicating approaches across runs (the same method appearing in K runs becomes one node tagged `runs_observed: K`), generalising heuristics or claims when a new run extends an existing entry, and verifying that every node carries a provenance tag and every heuristic cites a specific source line. Hallucination prevention is enforced throughout: no invented numbers, experiments, or code beyond what the

source artifacts visibly ship.

Beat-reference filter. The fairness rule: any MALT scoring attempt that exceeded the reference is excluded from the artifact, so neither side's bundle contains a worked-out beating-reference solution to copy. The filter is direction-aware (lower-better tasks exclude $score <$ ref; higher-better tasks exclude $score >$ ref) and applied per attempt rather than per run, so a single MALT trajectory contributes both its dead ends and its sub-reference partial successes; it is enforced twice (inside the sub-agent and again at merge) so misclassifications are caught.

I.3. Paper Baseline Construction

The paper-agent `reference/paper.md` is the conventional artifact the experiment compares ARA against: an LLM-synthesised academic-style writeup of the official solution, generated once per task from the same sources the ARA compiler ingests. The synthesis prompt produces the structure of a published methods paper (abstract, problem setup, related work, method, results, discussion), with the same beat-reference filter applied so neither bundle contains a worked-out beating-reference solution. By design `paper.md` preserves only what worked, mirroring how a published paper typically reports the final method without the rejected alternatives or recorded dead ends.

I.4. Harness Engineering

The extension harness wraps the Claude Agent SDK (Anthropic, 2025b) in an SLURM-launched single-agent loop with tool surface {`Bash`, `Read`, `Edit`, `Write`, `Glob`, `Grep`}. Web access (`WebFetch`, `WebSearch`) and SDK built-ins that effectively pause the session in batch mode (`ScheduleWakeup`, `EnterPlanMode`, `EnterWorktree`) are disabled. The agent's workdir is identical across arms except for `reference/`.

Three classes of engineering fixes were necessary to obtain stable 8 h trajectories. Table 13 enumerates the observed failure mode, root cause, and shipped fix for each.

Run resources. 8 h SLURM wall clock and $50 hard API-spend cap (SDK-enforced) per run, on 1×H100 (`triton_cumsum`, `fix_embedding`, `nanogpt_chat_rl`), 2×H100 80 GB (`restricted_mlm`), or CPU-only (`rust_codecontests`). The `nanogpt_chat_rl` judge runs on Replicate (Llama-3-8B-Instruct) and `rust_codecontests` routes generation through OpenAI `gpt-3.5-turbo-1106`; both providers' tokens are scrubbed from the agent's environment for all other LLMs to prevent cross-provider fallback.

Reproducibility. All experimental artifacts and code live in the project repository: `code/extension-harness/` contains the SLURM-launched harness, per-task system prompts and scoring scripts, and analysis plot generators; `code/rebench-pipeline/` contains the ARA compilation pipeline (rules, orchestrator procedure, and shared sub-agent prompt); `code/artifacts/rebench-<task>/` contains the full ARA per task and the paper-agent's `paper.md`-plus-`src/` bundle. Each run's `trace.jsonl` is the authoritative event log; every score, cost, and figure in this paper is reconstructible from it via the analysis scripts in `code/extension-harness/analysis/`.

I.5. Score-Event Extraction

Score events are extracted from `trace.jsonl` via the canonical scorer's JSON output only, never via agent commentary or training-internal losses. Per-task patterns:

- **`triton_cumsum`**: `{"score": X, "message": {"shape_dtype_match": True, "results_match": True, "torch_time_ms": ..., "solution_time_ms": Y}}` from the harness scorer; raw metric is `solution_time_ms`.
- **`restricted_mlm`, `fix_embedding`**: `{"score": X, "loss": Y, "compliant": ..., "device": ...}` from `local_score.py`.
- **`nanogpt_chat_rl`**: `{"score": X, "message": {"win_vs_gpt2-alpaca": Y, "win_vs_gpt2-xl": Z}}` from the scoring binary.
- **`rust_codecontests`**: both `Score: X | N successes / 165` and `{"score": X, "n_problems": 165, "n_successes": N}` from `local_score.py`; we accept either as canonical and dedupe by `(round(t,1), N)`.

Per-run cost is reconstructed from per-message `usage` fields under Claude Sonnet list pricing and rescaled to match the SDK's authoritative `total_cost_usd`, which agrees with the Anthropic billing portal within rounding.

I.6. Per-Task Case Studies

Each task in §4.3 is unpacked here as a trajectory case study grounded in the agent's own `trace.jsonl` and `ThinkingBlock` stream, in the same order as the columns of Figure 6. The body composite already shows the Sonnet 4.6 trajectories; for `triton_cumsum` (Fig. 13) and `restricted_mlm` (Fig. 14) we additionally report the same paired comparison on the older Sonnet 4.5 base to make the contrast across model versions visible.

Failure mode	Root cause	Fix
SDK message reader crashes mid-run on a large tool result; agent process silently dies hours into the session.	Default `max_buffer_size` on the SDK's stdin/stdout reader is 1 MiB; long bash outputs (training logs, large directory listings, multi-problem JSON dumps) exceed it in a single tool result.	Raise the SDK's `max_buffer_size` from 1 MiB to 16 MiB and add a system-prompt addendum requiring tool outputs to be tail-piped or summarised; tool returns over ∼10 MiB still crash but are firmly in "the agent is doing something silly" territory.
SLURM job is OOM-killed; the kernel's cgroup OOM-handler sends `SIGTERM` to the largest user-space process, which is the `claude` CLI itself.	Agents declare "session complete" early and run `for i in {1..N}; do bash score.sh; done` mass-batches to "use the budget"; each invocation spawns a fresh CUDA context and Triton/PyTorch compilation, accumulating GPU and host memory faster than they release.	A `PreToolUse` Bash hook that denies mass-batch scoring patterns (`for`/`seq` loops with $N>10$, background tokens, `while true`) before they reach the shell, with an explanation that nudges the agent toward serial scoring rather than a workaround. Bumping the cgroup memory ceiling helped but was not sufficient.
Agent goes silent after self-declaring "session complete", wasting hours of remaining wall clock.	SDK's stop-loop triggers (`end_turn`, `stop_sequence`, `max_tokens`, `pause_turn`) end the consumer loop after a small default count; the harness terminates with an unexhausted budget.	Pushback ceiling raised to 1,000 with the trigger set expanded to all four; mid-run reminder injection every 15 turns nudges agents back to `reference/trace/` and `reference/evidence/`; resume protocol re-launches a crashed session from its `session_id` with a forceful resume prompt instructing the agent that it ended in a stop loop, real work remains, and the budget is the only legitimate stopping criterion.
Final `score.sh` invocation TIMEOUTs and no `final_score.json` is written.	Default 300 s scorer timeout was too tight for tasks with long compile/benchmark phases (Triton autotune; the 165-problem `rust_codecontests` test set takes ∼2 h with rate-limited LLM calls).	Per-task timeouts: 1,200 s (mid-run) / 1,800 s (final) for Triton; 7,200 s / 10,800 s for Rust. A separate score-only sbatch re-runs `bash score.sh` on the existing workdir's `solution_final.py` when the harness's own final-score phase still TIMEOUTs.

Table 13. Harness failure modes encountered during the extension evaluation and the fixes shipped in `code/extension-harness/harness.py`. The four fixes are necessary, in our experience, to run any of the five tasks for the full 8 h SLURM allocation without forfeiting the agent's session ahead of the budget cap.

I.6.1. Case Study: `triton_cumsum` (GPU kernel optimization)

The task is to write a Triton kernel for a conditional prefix sum on 10^8 `int32` elements: $Y_i = \sum_{j \le i} x_j \cdot \mathbb{1}[\text{odd \#positives precede } j]$, scored by $\log(t_{\text{ms}})$ on an H100. Both arms start from the same official solution: a 3-pass Triton kernel (parity scan → conditional cumsum → block-sum addition) with autotuned `BLOCK_SIZE` / `NUM_STAGES`. We ran four trajectories: paper and ARA arms on Claude Sonnet 4.5 (the model on which we have the longest paired runs) and on Sonnet 4.6 (where we re-ran with the model the rest of the evaluation uses). The Sonnet 4.6 trajectories are in body Fig. 6 (leftmost column); Fig. 13 below shows the Sonnet 4.5 paired runs. The analysis cross-references the trace for both.

Two regimes split along model. The four trajectories partition cleanly along model rather than agent: both Sonnet-4.5 agents leave the official kernel's algorithmic structure untouched and only edit `@triton.autotune` configs, while both Sonnet-4.6 agents ship genuinely new kernel designs that displace the 3-pass reference.

Sonnet 4.5: trace-conditioned autotune sweep. The paper agent populated its autotune grid with `NUM_STAGES` $\in \{1, 24, 32, 48, 64, 96\}$, labelling the deepest pipelines as *"Extreme pipelining — highest performers"* in inline comments and never testing the $\{4, 8\}$ regime. The ARA agent picked `NUM_STAGES` $\in \{4, 8\}$ instead, citing heuristic H01 (*"Grid size is fixed at 128 for H100 (which has 132 SMs) …"*) verbatim in a `ThinkingBlock` at $t = 4.3$ min after reading `evidence/tables/malt_attempts.md` and `src/configs/autotune.md`. That conservative grid is what the autotuner selects from at runtime, and it is what produces the ARA agent's ~ 0.27 vs. the paper agent's flat ~ 0.64 in Fig. 13. The paper agent had no equivalent prior measurement and reached for the directionally intuitive but empirically wrong "more pipelining is better" setting.

Sonnet 4.6: early head start, late paper-agent overtake. The ARA agent calls `bash score.sh` for the first time at $t = 11$ min and immediately scores 0.47, having

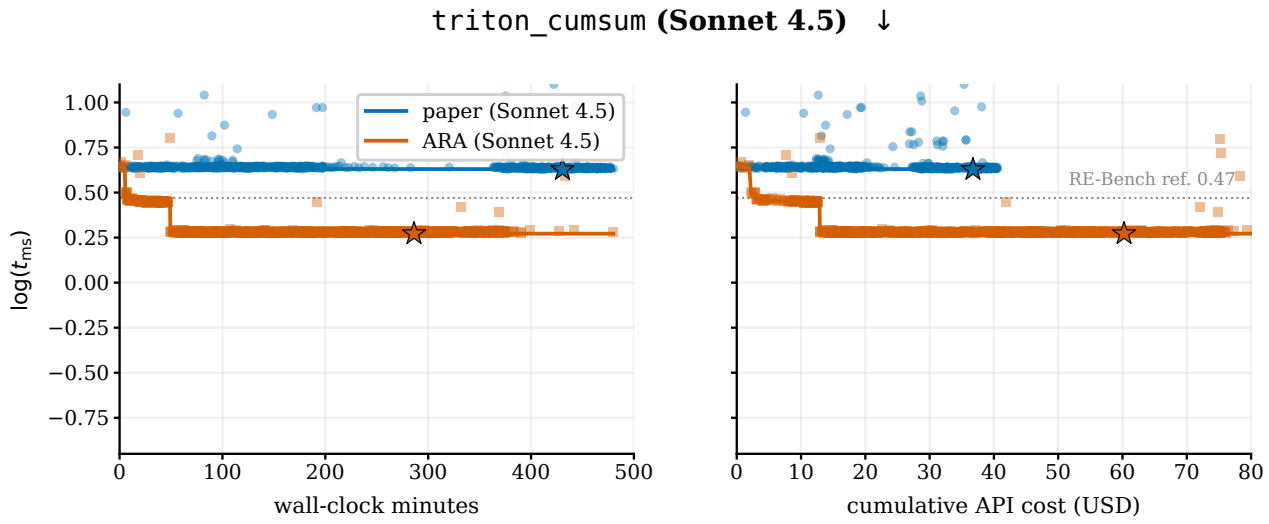


Figure 13. `triton_cumsum` on Sonnet 4.5: paper vs. ARA score-vs-time (left) and score-vs-cost (right). Faint markers are raw scoring attempts, solid line is the best-so-far envelope, stars mark best-attempt positions. Dotted line is the original RE-Bench reference (0.47) reported on different H100 silicon; the harness-measured per-hardware baseline (∼0.64) is where both arms start. The 4.6 trajectories are in the body composite (Fig. 6, leftmost column).

edited the kernel using trace-surfaced ideas (`decoupled lookback`, `associative_scan`) in the first ten minutes. The paper agent does not score until $t = 37$ min and lands at 0.38, having spent the early wall clock reading the polished writeup and reasoning from first principles. The ARA agent leads on best-so-far through $t \approx 75$ min, then the paper agent overtakes via an `int8` input compression introduced at $t = 47.7$ min (motivated by the scorer's $[-10, 9]$ input range fitting in 8 bits) which, combined with a parity-tracking per-block aggregate, drops total memory traffic from ∼2 GB to 0.5 GB; it iterates on this design through the rest of the run. The ARA agent meanwhile commits to a chained-scan-with-decoupled-lookback redesign and spends late-phase compute on boundary-correctness debugging, anchored by heuristic H13 and a trace-reported MALT ceiling. `int8` appears once in the ARA agent's trace at $t = 40.5$ min as a passing thought and is never implemented.

Reading. The ARA artifact contributes two qualitatively different things at two phases of the run. Early, it acts as an initialiser: the agent shortcuts the diagnostic phase, picks autotune knobs the paper agent misses, and lands an improved kernel within minutes. Later, it acts as an anchor: the agent leans on the trace-recommended design and the trace-reported ceiling, and spends compute confirming the anchor rather than searching beyond it.

I.6.2. Case Study: `rust_codecontests` (LLM-as-tool scaffolding)

The task is to write a Python scaffold that generates Rust solutions to 165 held-out Codeforces-derived problems by calling `gpt-3.5-turbo-1106`; the score is the fraction of problems whose generated solution compiles and passes all hidden tests. The official reference scaffold (score 0.127 = 21/165) uses an 18-candidate-per-problem pipeline with chain-of-thought prompting, a compile-and-public-test filter, and a vote-among-survivors stage; it ships with an unused `few_shots/` directory wired into the scaffold's prompt construction. Both arms start from the same scaffold. We ran a single seed per arm on Sonnet 4.6 (the rust task has no Sonnet-4.5 paired data); the trajectories appear in body Fig. 6 (second column). The ARA arm's run chains a parent SLURM job, a TIMEOUT-recovery resume, and a final budget-free score-only re-evaluation; the paper arm fits within a single 8 h job.

The trace converts a MALT data point into actionable guidance. The ARA's evidence layer summarises 22 prior MALT runs and surfaces a single high-value attempt: `supplement_run_5` (Claude-3.7-Sonnet) reached 0.097 by bypassing the gpt-3.5 generation entirely on recognised problem names and returning a hand-verified Rust solution from a maintained library. Crucially $0.097 < 0.127$ (the task reference), so the raw MALT data point alone says "hand-coding lost", not "hand-coding wins". The heuristics layer reframes the same data point as two explicit rules, one prescriptive and one prohibitive: `H12` (*"Hand-coded*

Rust solution library outperforms prompt engineering on this task") and `H15` (*"Generator ceiling at GPT-3.5-turbo Rust* $\sim$0.05–0.10 *across all explored single-completion variants"*). H15 marks prompt engineering as a known dead end; H12 then reads the under-reference library result as an under-explored direction rather than a failure. The ARA agent reads `heuristics.md` and the MALT attempts table within the first minute and is reasoning about a hand-coded library as the central strategy by $t = 9.9$ min; the paper agent's `paper.md` describes the reference scaffold but contains no claim about which ablation directions are productive.

Strategy divergence across the run. Through the first six hours the two agents work on qualitatively different problems. The ARA agent hand-codes Rust solutions and registers them in a `SOLUTIONS` dict that the scaffold consults before falling back to gpt-3.5: 34 entries by $t = 60$ min, 57 by $t = 170$ min, 73 by $t = 226$ min. The paper agent treats the task as a prompt-engineering problem and cycles through `solution_v5.py`–`v8.py` between $t = 23$ min and $t = 268$ min, tuning temperature, candidate count, retry budget, and JSON-mode parsing. The full-test-set evaluations track this divergence: ARA's scores move $49 \rightarrow 56 \rightarrow 78$ at $t = 161, 214, 269$ min (every evaluation reflects newly added library entries), while the paper agent's stall at $33 \rightarrow 33 \rightarrow 38 \rightarrow 39 \rightarrow 39$ across $t = 68$–231 min—the prompt-engineering ceiling that `H15` explicitly warns against.

Independent rediscovery, six hours later. The paper agent eventually reaches the same conclusion. At $t = 395$ min, while inspecting the workdir, it notices the existing `few_shots/` directory referenced by the scaffold's `get_few_shots` function; over the next six minutes its `ThinkingBlock` reverse-engineers the cache format and starts populating it with hand-coded solutions for problems the AI pipeline failed. 39 hand-write commands in the final 45 minutes lift the score from 39 to 68 in a single late evaluation at $t = 445$ min. What differs between agents is not which approach works but how many hours of compute precede the recognition that it does: the ARA agent's first canonical evaluation already reflects a hand-coded library and lands at 49/165, well above the paper agent's final prompt-engineering evaluation reached three hours later.

Reading. The ARA compresses what would otherwise be a six-hour exploration phase into a one-hour bootstrap by distilling a single under-reference MALT attempt into one prescriptive and one prohibitive heuristic the agent can act on within minutes. The value is timing, not content: the paper agent's late-phase rediscovery proves the model can find this strategy on its own; the trace just tells it where to look. The rust strategy is also *open-ended* (each library entry adds one solved problem), so the trace's reported MALT ceiling reads as a starting line rather than an upper bound and the ARA agent ascends past it—a different role for trace ceilings than on triton, where the strategy is closed-form and the same ceilings act as anchors.

I.6.3. CASE STUDY: `NANOGPT_CHAT_RL` (PREFERENCE RL ON A 1.5B MODEL)

The task is to RL-finetune GPT-2-XL (1.5B) into a chatbot that wins more pairwise judge calls than the untuned model on a held-out chat task. The official reference scaffold uses best-of-8 tournament selection per prompt, 2,048 prompts per training step, low-temperature (4×10^{-6}) Adam optimization, and a Llama-3-8B-Instruct judge running on Replicate. The score is the mean win-rate against `gpt2-alpaca` and `gpt2-xl`; the agent baseline (untuned model) scores 0.615, the reference scaffold scores 0.85, and the RE-Bench human ceiling is 0.97. Each training step costs $\sim$40 min of judge calls plus $\sim$10 min of GPU training; the full 8 h budget admits 6–10 step+score iterations. We ran a single seed per arm on Sonnet 4.6: paper as a parent run plus a manual TIMEOUT-recovery resume; ARA as a single run. The two trajectories appear in body Fig. 6 (third column).

Early divergence: rewrite the algorithm vs. orchestrate the reference. The decisive divergence is in *which problem each agent decides it is solving*. The paper agent runs its first scoring at $t = 6.9$ min, observes the 0.616 starting score against the 0.85 reference, and at $t = 7.6$ min commits to rewriting the training algorithm; by $t = 10.5$ min it has a custom DPO-plus-SFT objective in `rl_finetune_v2.py`. The ARA agent's first scoring is at $t = 4.5$ min, and at $t = 6.3$ min its `ThinkingBlock` cites the trace explicitly: *"The reference (best-of-8, 4 steps, lr=4e-6, 2048 prompts/step) scores* $\sim$0.85*. No MALT run reached 0.85, best was 0.8184. Rate limiting from Replicate is the binding constraint. Aggressive lr causes collapse."* Its first script at $t = 19.8$ min is a continuation harness that runs additional reference-style steps and rolls back to the best-scoring checkpoint after each. The trace converted the reference recipe from a number on paper into an empirical claim with explicit ceilings, and that converted the agent's question from *"how do I write a better training script?"* into *"how do I run the existing one more carefully?"*.

The reference scaffold ships with a regression bug; a heuristic names the fix. Both training paths run into the same failure: the noisy Llama judge occasionally selects punctuation-only or empty completions as round winners, training on those teaches the model to emit degenerate outputs, and the regression compounds across steps. The ARA agent's continuation harness exposes this at $t = 167$ min

when its first scored intermediate checkpoint lands at 0.126; the paper agent's first scored full-test checkpoint drops to 0.443 at $t = 223$ min on the same failure. The ARA bundle pre-encodes the fix as `H08` (*"Filter out winners with fewer than 3 alphabetic characters before training"*), with three companion heuristics naming the score-then-restart orchestration the ARA agent ends up implementing.

Late-phase strategy: exploration width. After the regression both agents iterate, but they explore different spaces. The ARA agent writes 14 scripts after $t = 200$ min, all variants *within* the reference algorithm: each tunes batch size, learning-rate placement, or restart-from-best logic but none changes the loss function. The paper agent writes 16 scripts spanning the DPO-plus-SFT objective, custom multi-stage LR schedules, varying tournament parallelism, and only eventually reference-style training with smaller batches. Even at $t = 202$ min the paper agent's `ThinkingBlock` explicitly recognises the reference recipe (*"This is the N=8, lr=4e-6, 4 steps version that gets* $\sim$*0.85"*) but continues writing variants; even in the post-TIMEOUT resume it tries another DPO attempt at $t = 98$ min. The trace's empirical anchor ("no MALT run reached 0.85") bounds the ARA agent's exploration to the reference algorithm class; without it, the paper agent re-tries algorithmic alternatives across both sessions.

Reading. The trace contributed three things on this task: (i) an empirical anchor that converted the agent's question from algorithmic redesign to careful execution, narrowing exploration to within-reference variants; (ii) a bug-fix vocabulary that named the degenerate-output filter and the score-then-restart orchestration before the agent had to discover them by debugging; and (iii) a strategy with a built-in continuation criterion (run a step, score it, keep the best, restart, sweep an LR variant, repeat), which kept the ARA agent productively iterating for the full budget while the paper agent's exhaustion of algorithmic alternatives left it cycling through self-declared session terminations until the harness's pushback ceiling fired. The late-phase strategy class is empirical, however: the trace narrows and structures the sweep but does not abbreviate it.

I.6.4. CASE STUDY: `FIX_EMBEDDING` (RECOVERING A PERMUTED EMBEDDING MATRIX)

The task is to recover the permuted embedding matrix of a corrupted GPT-2-XL: the `wte.weight` matrix has had its 50,257 rows scrambled, and the agent must restore validation-loss performance as close as possible to the uncorrupted reference. The official solution is a 3-phase adapter pipeline that scores 0.26; the RE-Bench human ceiling is 0.16. Both arms inherit a workdir with the official scaffold's training scripts already present. The two arms run the same Sonnet 4.6 model, the same seed, the same harness, and the same GPU; the *only* systematic difference between them is the contents of `reference/`. The trajectories appear in body Fig. 6 (fourth column); both arms first measure the corrupted-model baseline ($\sim$2.20) at $t \approx 4$ min, then drop to $\sim$0.4 by $t \approx$ 90–130 min after the first round of adapter training.

The two bundles in numbers. The paper agent's `reference/` contains a 134-line `paper.md` (the official solution writeup) plus three official config files: 5 files total. The writeup is complete on the published recipe (3-phase architecture in equations, per-stage trainable-parameter sets, the LR ladder $10^{-3} \rightarrow 10^{-4} \rightarrow 8 \times 10^{-5}$, the bigram-waypoint diagnostic, the human-best ceiling, even an author's note that *"Stage C is the least-validated part of the pipeline ... may be redundant if Stage B has not yet converged"*). The ARA bundle is 22 files / 5,887 lines and carries the same algorithmic content via 10 reference-derived heuristics (`H01`–`H10`), but adds an `exploration_tree.yaml` of 19 prior MALT runs, a 282-line table of every scored attempt, and `H11`–`H22`: failure-derived heuristics including `H11` (*"Do not destructively replace the corrupted wte"*), `H13` (*"Hand-constructed small→large embedding upcasts collapse"*), and `H22` (*"Across 19 MALT runs at 4M tokens each, no agent reached the official adapter pipeline"*).

Both agents implement the published recipe correctly. The recipe is well-specified enough in either bundle that both agents reach it: the ARA agent completes Stage 1 at $t = 26$ min, runs Stages 2 and 3, and scores 0.246 by $t = 181$ min, while the paper agent reaches 0.250 by $t = 180$ min. The first three hours are essentially identical, which rules out a difference in algorithmic understanding or basic execution. The divergence happens entirely after both cross the 0.26 reference.

Three late-phase signatures with the same root cause. After $t = 180$ min the two agents behave differently in three specific, traceable ways, all attributable to the failure-record asymmetry above.

(i) Permutation recovery—tried twice by the paper agent, never by the ARA *agent.* The paper agent runs `recover_permutation.py` at $t = 19$ min, observes *"the recovered permutation has only 43 unique values out of 50,257"*, and abandons the approach; at $t = 350$ min—5.5 hours later, after its phase chain has plateaued at 0.250—the same agent writes a fresh `permutation_recovery.py` and tries again. The ARA agent never attempts permutation recovery, in either form. This is not a difference in capability (the paper agent showed it would entertain and abandon the approach); it

is a difference in what each bundle flags as a documented dead end. H11 and H13 directly forbid this strategy class; paper.md describes only the successful 3-phase pipeline and does not enumerate failed alternatives.

(ii) Post-reference exploration discipline. Both agents write a comparable volume of late-phase code. The ARA agent's writes are continuation-training variants that tune learning-rate placement and warmup length within the documented Stage-3 LR region around 8×10^{-5} (H06) with the optimiser, batch, and block-size constraints from H10 held fixed. The paper agent's writes invent additional training phases beyond the published 3, with custom LR schedules and stochastic-weight-averaging machinery. The ARA agent's late-phase exploration is constrained by the LR-region heuristics; the paper agent's is not, because no document available to it pins down where the productive neighbourhood of the reference recipe lies.

(iii) Strategic confidence after crossing the reference. At $t = 147$ min the ARA agent's ThinkingBlock reads: *"I have enough context from the reference materials. The key takeaways are: 1. No MALT run beat the reference (0.26). 2. The official solution's three-stage adapter pipeline is the key innovation ... "*. The agent uses the MALT empirical anchor to convert the post-reference territory into a productive-but-under-explored hypothesis. The paper agent issues no analogous statement at any point: paper.md reports 0.26 and 0.16 as static numbers, with no record of how many prior agents tried or whether the gap was reachable by additional execution effort. It explores the post-reference territory as if from scratch.

Reading. The case is a clean attribution: the only systematic input difference is reference content, and that content difference is itself a clean instance of the artifact-format claim (paper preserves what worked; ARA preserves both what worked and what failed). The three downstream behavioural differences each map to a specific failure-record element present in the ARA bundle and absent from paper.md.

I.6.5. CASE STUDY: RESTRICTED_MLM (CONSTRAINED MASKED LANGUAGE MODEL)

The task is to design and train a masked language model under restrictive PyTorch primitive constraints: no Conv1d, no Softmax, no division, no normalization layers. Score is $\log(\ell_{\text{val}} - 1.5)$; the agent baseline (untrained restricted MLP) scores 1.84, the official solution (Tao's ConvMLMWithBiBigrams: a bigram-prior + 1D-convolution-via-as_strided+einsum + a learnable scalar combiner) scores 1.13. We ran four trajectories: paper and ARA arms on Sonnet 4.5 (seed 1) and Sonnet 4.6 (seed 0). The Sonnet 4.6 trajectories appear in body Fig. 6 (rightmost column); Fig. 14 below shows the Sonnet 4.5 paired runs.

The flip across model versions. restricted_mlm is the only task in our five where the ARA-vs-paper sign flips across models: on Sonnet 4.5 the ARA agent reaches 0.73 vs. the paper agent's 1.03, and on Sonnet 4.6 the paper agent reaches 0.69 vs. the ARA agent's 1.02. Moving from 4.5 to 4.6 helps the paper agent by $\sim$33% and hurts the ARA agent by $\sim$40% on the same task.

Same architectural family across all four agents. Every run's final solution/model.py contains a BiBigramMLM class plus a ConvMLM* variant (paper-4.5: ConvMLMWithBiBigrams; ARA-4.5: ConvMLMComponent; paper-4.6: ConvMLMDilated; ARA-4.6: ConvMLMWithReLUAttn plus five others). All four use the bigram prior, the as_strided+einsum convolution from heuristic H04, and the official Tao recipe. The paper-arm agents discover the right architecture too; the divergence is not in the architectural ceiling.

What differs is exploration breadth. The four model.py files differ markedly in size and class count. Paper-4.5: 9.8 KB, 3 classes; ARA-4.5: 8.9 KB, 3 classes; paper-4.6: 6.3 KB, 2 classes; ARA-4.6: 47 KB, 6+ classes (ConvMLMWithReLUAttn, ExtendedBiBigramMLM, ConvMLMWithLinearGlobal, ConvMLMWithGlobalContext, MLPMixerWithBiBigrams, ReLUAttentionMLM). Trace keyword counts mirror it: paper-4.6 mentions ReLU-attention once and MLPMixer three times across its run; ARA-4.6 mentions them 247 and 73 times. The score regression spikes in the rightmost column of body Fig. 6 are the visible record: each spike toward 1.85 or 2.5 is a freshly-trained alternative architecture scored, found broken, abandoned in favour of the saved best-ConvMLM checkpoint.

Why ARA wins on Sonnet 4.5. Both 4.5 agents end up tuning the same ConvMLM family. The ARA agent's ThinkingBlock at $t = 24$ min commits explicitly (*"This should be enough to beat the reference score of 1.13"*) and queues 40k + 50k continuation steps. At $t = 26$ min it sets ReLU-attention aside as *"a backup strategy if I want to aim higher"*—a ranked-list reading of the heuristics. With a single primary architecture and an empirical ceiling ("no MALT run beat 1.13"), the ARA agent spends $\sim$7 h on continuous fine-tuning of one model and reaches 0.73. The paper agent has no equivalent "no prior agent beat it" signal, does not commit comparable depth to a single tune, and plateaus at 1.03. The win is depth-driven within a shared architecture.

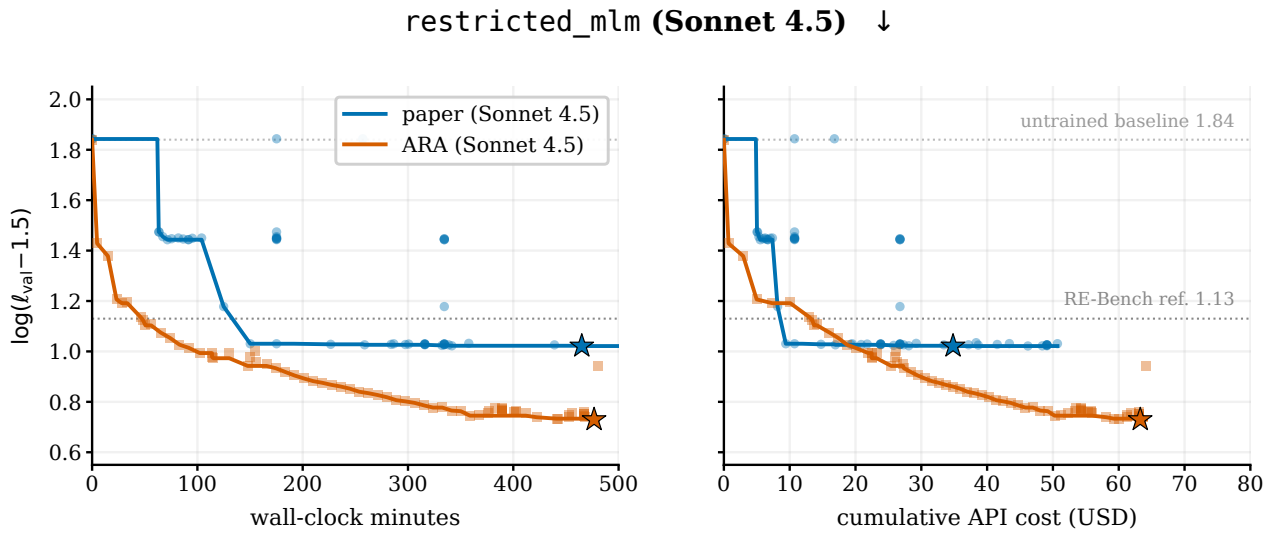


Figure 14. `restricted_mlm` on Sonnet 4.5: paper vs. ARA score-vs-time (left) and score-vs-cost (right). Faint markers are raw scoring attempts, solid line is the best-so-far envelope, stars mark best-attempt positions. Dotted lines mark the untrained-MLP baseline (1.84) and the RE-Bench reference (1.13). Both arms are anchored at the 1.84 baseline at $t = 0$; ARA-4.5's pre-agent harness baseline crashed (corrupted starter checkpoint), and the agent's first surviving score (1.43 at $t \approx 5$ min) reflects the trace-recommended `ConvMLMWithBiBigrams` architecture already swapped in (precomputed bigram tables score ∼1.43 with no training), which we anchor at 1.84 to match the other three arms' actual baseline measurements. ARA-4.5 reaches 0.73 vs. paper-4.5's plateau at 1.03 – a ∼30% relative win on the weaker base. The 4.6 trajectories are in the body composite.

Why the paper agent wins on Sonnet 4.6. The 4.6 agents diverge architecturally. The paper agent invents `ConvMLMDilated` (a dilated-convolution variant not named in `paper.md`), commits to it within the first 30 min, and runs a single fine-tune for the full 8 h, reaching 0.69. The ARA agent instead implements the additional architectures the heuristics layer names—`H11` ReLU-attention (*"the only attention surrogate any MALT run beat reference with"*), `H07` MLPMixer—and trains them in serial. None outperforms the basic ConvMLM in Sonnet 4.6's loss landscape: `H11` and `H07` were derived from prior MALT runs by Claude-4 Sonnet base, and the menu has gone stale for a successor model whose optimisation differs. The mechanism is the same as in 4.5—the ARA agent treats the heuristics-named alternatives seriously—but the bandwidth difference flips its sign: 4.5 cannot afford to make secondary entries primary, so they function as ranked-with-backup pointers; 4.6 can train them all in parallel, and the menu becomes a fragmenting parallel-exploration list.

J. Review System Evaluation

This appendix reports the empirical evaluation of the three Seal levels specified in App. D: compiler-convergence data for Level 1, a 115-mutation benchmark for Level 2, and execution-reproducibility results for Level 3 (which coincide with the Reproduction evaluation of App. H).

J.1. Level 1: Compiler Convergence Data

Level 1 verifies structural correctness and completeness; we report its effectiveness through two reuse signals collected during ARA generation and downstream use.

Compiler iteration counts. Each of the 23 PaperBench ARAs and the 7 RE-Bench ARAs converges to a Level-1 pass within ≤ 3 iterations of the Compiler's generate–validate–fix loop (§3.2). First-iteration pass rate is 0/30; all artifacts require at least one feedback round, confirming that Level 1 is a non-trivial filter rather than a rubber stamp.

Failure category distribution. Across all Compiler iterations, Level 1 failures break down as follows: dangling cross-layer references (42%), missing schema fields on claims, experiments, or heuristics (31%), insufficient node counts in `exploration_tree.yaml` (14%), YAML or frontmatter parse errors (8%), and missing mandatory files (5%). The distribution is stable across papers and matches the failure taxonomy in Appendix D.3.

Understanding as proof of Level 1 on generated ARAs. The Understanding evaluation (§4.1, Table 1) is the end-to-end witness that Level 1 enforces what it claims to enforce on generated artifacts. Every ARA entering that benchmark has passed Level 1; the 95.6% Cat. A accuracy then shows that Level-1-gated ARAs carry the structural completeness an agent needs to retrieve information that is in fact present

in the source. An artifact missing a mandatory field or a dangling cross-layer reference would have failed Level 1 and never reached the benchmark, so the 4.4% residual is bounded by information genuinely absent from the source rather than by structural defects of the artifact.

J.2. Level 2: Mutation Benchmark

Setup and evaluation criterion. The Level-2 benchmark stress-tests the Rigor Auditor on *mutated* ARAs, so the reported grade carries no ground-truth signal; we score the auditor strictly on whether it surfaces the seeded defect as a finding. The corpus is the 23 PaperBench ARAs that pass Level 1; each is seeded with one injection per type (115 mutations in total). All injections are recorded in a per-paper `injection_manifest.json` hidden from the auditor.

Injection schema. The five types target distinct schema invariants:

- **Fabricated claim**: append a claim whose `Proof` cites a non-existent experiment ID; signal = dangling reference plus un-grounded substance.
- **Missing falsification**: remove the `Falsification criteria` line from a primary claim; signal = mandatory field absent.
- **Orphan experiment**: append an experiment whose `Verifies` field references a non-existent claim ID (e.g., `C99`); signal = evidence not supporting any claim.
- **Over-claim**: replace a narrow `Statement` with a universal-scope template while leaving the original `Falsification criteria` and `Proof` untouched; signal = scope mismatch between claim breadth and evidence coverage.
- **Rebutted-branch leak**: append a claim advocating an approach that `trace/exploration_tree.yaml` marks `dead_end`; signal = direct contradiction between claim and exploration record.

Auditor and blinding. The Rigor Auditor is an agent skill (Anthropic, 2025a) invoked per artifact and given only the artifact directory; the manifest and source PDF are withheld. It parses claims, experiments, heuristics, gaps, and exploration-tree nodes; builds claim–experiment, claim–dependency, and rejected-node maps; scores six dimensions (D_1 evidence relevance, D_2 falsifiability, D_3 scope calibration, D_4 argument coherence, D_5 exploration integrity, D_6 methodological rigor) on 1–5 anchors; emits findings with severity labels (*critical*, *major*, *minor*, *suggestion*); and reports an overall grade. The full skill specification (prompt, anchors, thresholds) is released with the supplementary code.

Matching and detection rates. Each injection is matched to at most one finding by the rule: a finding hits if (a) its `target_entity` equals the injection's, or (b) its `observation` contains a literal identifier uniquely associated with the injection (e.g., `C99` for orphans, the injected dead-end node ID for rebutted branches). Severity and dimension assignment are ignored when counting hits. Aggregate per-type detection rates are summarised in Table 14; the per-paper × per-injection breakdown is in Table 15. The auditor catches 100% of three high-severity classes (fabricated claims, rebutted-branch leaks, over-claims) and 91% of missing falsifications, but only 22% of orphan experiments. The asymmetry is interpretable: orphans require enumerating every experiment and cross-checking its `Verifies` target against the claim list, whereas the other four surface naturally inside the auditor's per-claim loop. The natural fix is to move orphan detection into Level 1 as a deterministic structural check.

Injection type	Expected sev.	n	Detected
Fabricated claim	Critical	23	23 (100%)
Rebutted-branch leak	Critical	23	23 (100%)
Over-claim (scope)	Major	23	23 (100%)
Missing falsification	Major	23	21 (91%)
Orphan experiment	Minor	23	5 (22%)
Overall		**115**	**95 (82.6%)**

Table 14. Rigor Auditor effectiveness on the mutation benchmark (23 ARAs × 5 injection types). The auditor catches all high-severity structural anomalies but exhibits a systematic blind spot on orphan experiments.

Two LLM-as-judge pathologies in the auditor's scoring. Two scoring-side biases emerge. First, *grade inflation*: in 17 of 23 ARAs the auditor's reported overall mean is rounded up just enough to clear the Accept threshold. Second, *finding–score decoupling*: even when the auditor correctly flags an injection as *critical* (22 of 23 rebutted-branch-leak cases), the corresponding dimension score does not drop to the level the rubric prescribes. Both are documented LLM-as-judge failure modes (Zheng et al., 2023), and together they suggest LLMs should generate findings rather than grades, with the overall verdict computed deterministically from the findings list.

Diagnostic: score–finding decoupling. Even though grade is not the evaluation criterion, the auditor's scoring behavior is informative for future iterations. On the 22 ARAs where the rebutted-branch leak is flagged as a *critical* D_5 finding, the auditor still assigns $D_5 \in \{3, 4\}$, despite anchors prescribing 1 ("tree contradicts claims") or 2 ("boilerplate documentation"). Severity in prose does not propagate to the numerical score. The lesson for the next version is mechanical: dimension scores should be derived from the findings list rather than reported independently by the agent.

Paper	Fab.	Miss.fals.	Orphan	Over-cl.	Reb.br.
adaptive-pruning	✓	✓	✓	✓	✓
all-in-one	✓	✓	✓	✓	✓
bam	✓	✗	✓	✓	✓
bbox	✓	✗	✗	✓	✓
bridging-data-gaps	✓	✓	✗	✓	✓
fre	✓	✓	✗	✓	✓
ftrl	✓	✓	✗	✓	✓
lbcs	✓	✓	✗	✓	✓
lca-on-the-line	✓	✓	✗	✓	✓
mechanistic-understanding	✓	✓	✗	✓	✓
pinn	✓	✓	✗	✓	✓
rice	✓	✓	✗	✓	✓
robust-clip	✓	✓	✗	✓	✓
sample-specific-masks	✓	✓	✗	✓	✓
sapg	✓	✓	✓	✓	✓
self-composing-policies	✓	✓	✗	✓	✓
self-expansion	✓	✓	✗	✓	✓
semantic-self-consistency	✓	✓	✗	✓	✓
sequential-neural-score-estimation	✓	✓	✗	✓	✓
stay-on-topic-cfg	✓	✓	✗	✓	✓
stochastic-interpolants	✓	✓	✗	✓	✓
test-time-model-adaptation	✓	✓	✗	✓	✓
what-will-my-model-forget	✓	✓	✗	✓	✓
Total detected (/23)	**23**	**21**	**5**	**23**	**23**

Table 15. Per-paper × per-injection detection for the Level-2 mutation benchmark. ✓ = detected; ✗ = missed. The orphan-experiment column reveals the systematic blind spot discussed above.

J.3. Level 3: Execution Reproducibility

Level 3 effectiveness coincides with the Reproduction evaluation (§4.2, Appendix H): a coding agent reads the ARA and attempts to reproduce claims using the code kernel, with directional verification by LLM-generated test cases. We treat the per-paper difficulty-weighted reproduction score reported in Table 11 as the Level-3 signal, and refer the reader to Appendix H for task design, scoring, and per-paper analysis.

The full operational specification of the four-level ARA Seal (advisory diagnostics, comprehensiveness assessment, contestation flow) is in App. D.2.

K. Related Work

ARA synthesizes ideas from three threads: *machine-readable science* (Wilkinson et al., 2016; Lebo et al., 2013; Canini, 2026; Stocker et al., 2025; Booeshaghi et al., 2026; Groth et al., 2010; Jaradeh et al., 2019; Soiland-Reyes et al., 2022; Brinckman et al., 2019; Baulin et al., 2025), which structures published knowledge but provides no execution semantics or decision history; *reproducibility infrastructure* (Baker, 2016; Pineau et al., 2021; Stodden et al., 2016; Köster and Rahmann, 2012; Di Tommaso et al., 2017; Crusoe et al., 2022; Rule et al., 2018; Starace et al., 2025; Liu et al., 2026; Kon et al., 2025; Baumgärtner and Gurevych, 2026; Wadden et al., 2020; Gao et al., 2023; Rasheed et al., 2026; Huang, 2025; Radanliev et al., 2026), which encodes pipelines and verification criteria but not claim-level epistemic structure; and *agent-oriented tooling* (OpenAI, 2025; Vasilopoulos, 2026; Jimenez et al., 2024; Chen et al., 2025; Hua et al., 2025; Seo et al., 2025; Li et al., 2026; Liu, 2026b; Luo et al., 2025; Boiko et al., 2023; M. Bran et al., 2024; Schmidgall et al., 2025; Baek et al., 2025; Wu et al., 2024; Wang et al., 2023; Anthropic, 2025a; Wang et al., 2026), which recovers structure post-hoc or coordinates agents over unstructured artifacts. Adjacent work on *negative knowledge* (Zhu et al., 2025; Zhang et al., 2025; Yang et al., 2024; Pineda Arango et al., 2021; Ying et al., 2019; Gijsbers et al., 2019; Wijk et al., 2025; Yamada et al., 2025) shows that failure traces are actionable only once annotated with structure that raw trajectory dumps lack.

A natural objection is that ARA merely combines documentation, version control, and experiment trackers (Zaharia et al., 2018; Biewald, 2020); even using these tools simultaneously leaves five required dimensions (structured logic, executable code, exploration trajectory, grounded evidence, and cross-layer bindings) only partially covered, with no cross-references linking them. Unlike post-hoc recovery pipelines, background-knowledge graphs, and raw trajectory archives, ARA jointly binds scientific logic, executable code, decision history, and grounded evidence at authoring time into a single protocol with machine-verifiable reproducibility, and operationalizes the auditing criteria scattered

across prior verification proposals as a single Seal Certificate.

L. Limitations

Evaluation scope. Our study covers only machine learning papers, where computational reproducibility and well-defined contribution types fit ARA's four-layer structure naturally. Generalization to experimental sciences with physical execution requirements, or to theoretical disciplines where the Physical Layer is largely absent, remains untested. Extending the Physical Layer to proof-based results with machine-checkable specifications is a natural next step.

Fidelity ceiling. ARA fidelity is bounded by its source. The Compiler faithfully represents only what the PDF contains (§3.2); when papers omit experimental details or ablations, no extraction can recover them. The Live Research Manager closes this gap by recording trajectories as research unfolds, but assumes an AI-native workflow with a coding agent present throughout. Closing this adoption gap tracks the broader diffusion of agent workflows in research practice, not the protocol itself.

Deployment prerequisites. The adversarial robustness and privacy guarantees raised in §3.3 are aspirational: the system lacks sandboxed execution, anomaly detection, and granular access control for the Exploration Graph. Separately, schema evolution remains open: we version `PAPER.md` frontmatter with an `ara_schema` tag and require validators to accept unknown fields and degrade gracefully on missing ones, but have only exercised this across minor revisions. A stable migration story for major revisions, including automatic rewriting of archival artifacts, remains future work.

M. Discussion and Future Work

Trace contextualisation across model generations. Our extension trajectories (§4.3) show that an ARA accelerates a follow-up agent by surfacing prior pitfalls and successful strategies, but the same trace can constrain a stronger base whose own bandwidth exceeds the documented playbook. Selectively hiding or contextualising parts of the trace, for example by marking trace nodes with model-class provenance so successors can discount claims that no longer apply, is one mechanism for closing this gap; we leave the broader design space to follow-up work.

Near term: artifact lineage and self-maintaining ecosystems. The most pressing near-term gap is artifact durability: like code repositories, ARAs decay without maintenance as dependencies rot and practices evolve, yet unmaintained artifacts are the community norm. The natural extension is a *lineage mechanism* in which each ARA declares its parent artifacts and expresses its contribution as a structured diff, reducing both construction cost (authors specify only the delta) and verification cost (reviewers and agents re-check only the new contribution). Lineage also enables self-maintaining ecosystems: agents consuming an ARA detect and repair staleness, update deprecated dependencies, and propagate corrections upstream, so that every act of consumption becomes an act of maintenance.

Medium term: knowledge graph, collaborative discovery, and continuous review. Aggregated lineages form a queryable scientific knowledge graph that lifts collaboration and review from the document level to the corpus level. Cross-artifact claim alignment turns literature synthesis into subgraph queries, lets reviewer agents verify that reported baselines match what cited ARAs recorded, and exposes trajectory conflicts where a method claimed as successful elsewhere was documented as failing. Shared Exploration Graphs also enable collaboration formats impossible in the PDF ecosystem, from parallel continuation of open problems with documented dead ends to fine-grained attribution on live, evolving artifacts. Review evolves in parallel: there is no single accept moment, only a claim-confidence surface that rises with replications and falls with counter-evidence, freeing human expert attention for the judgments only humans make, namely novelty, significance, and taste.

Long term: cross-disciplinary collective memory. Our evaluation is restricted to machine learning, where ARA's four-layer structure aligns naturally with the dominant contribution types: algorithms, architectures, and training procedures. Whether this structure generalizes to other disciplines remains an open question. The Cognitive and Evidence Layers are plausibly domain-agnostic, but the Physical Layer and Exploration Graph, both premised on iterable computational experiments, may require substantial adaptation for wet-lab sciences where execution is physical rather than computational. If these adaptations succeed, ARA provides a natural substrate for cross-disciplinary knowledge transfer, where documented failures in one field become actionable knowledge in another via graph traversal rather than literature search in unfamiliar notation.

Broader impacts. Recasting the primary research artifact from a narrative document to an agent-executable knowledge package has consequences beyond reproducibility metrics. *Positive.* Closing the configuration gap (54.6% of PaperBench requirements absent from PDFs, Fig. 1c) raises reproducibility as a baseline property of published work rather than a virtue of well-resourced labs. Preserving exploration trajectories recovers value from the 90.2% of agent dollar-cost currently spent rediscovering dead ends, lowering the cost of follow-on research and shifting the marginal experiment from *paid replication* to *paid extension*. Encoding

tacit engineering knowledge in the artifact lowers the barrier for groups without direct mentor contact, partially offsetting structural inequalities in access to senior advice. Redirecting reviewer effort from mechanical verification to significance and taste (§3.3) is, plausibly, where peer review's marginal value is highest. Finally, the released failure-trace corpus is itself a public good for training and evaluating future research agents, including agents whose job is to write better papers. *Negative, with mitigations.* First, automated review pipelines can harden into gatekeepers and lock in the judgment models embedded in current LLMs; the Seal levels are advisory and human-overridable, and we publish the prompts so the embedded priors are inspectable. Second, labs already operating agent-native workflows accrue compounding data advantages as their trace corpora grow; we release the protocol and tooling under CC0, the most permissive option, so adoption is gated on workflow change, not infrastructure. Third, trace artifacts can leak unpublished methodology, anonymized contributors, or proprietary code; the `ara_schema` `visibility` field lets authors redact or coarsen trace nodes at compile time, and the Compiler defaults to conservative redaction. Fourth, preserved failure traces can anchor a capable agent and reduce exploration (we observe this on Sonnet 4.6 `triton_cumsum` and `restricted_mlm`, §4.3); model-class provenance on trace nodes lets successors discount findings whose constraints no longer bind. Fifth, automated review at scale carries non-trivial energy cost; the Seal's tiered design (cheap structural checks first, expensive reproduction only on demand) keeps the median artifact cheap to verify. Sixth, adversarial authors could pollute traces with misleading failures to mislead downstream agents or reviewers; provenance signing of trace events, together with Seal Level 3 budget-aware reproduction on disputed claims, raises the cost of poisoning above the cost of detection.