Scientific-RAM: Research Agents Need Role-Aware Handoff Memory

Anonymous ACL submission

Abstract

LLM-based scientific agents increasingly decompose research workflows into specialized roles such as reading, planning, coding, evaluation, and reporting. This decomposition creates a fragile handoff problem: downstream agents must preserve validity-critical data, metric, artifact, and evaluator constraints that are often weakened by generic summaries or retrieval outputs. We formalize this setting as research handoff, where the downstream agent, action, evaluator, and execution budget are fixed, and only the transferred packet varies. We propose Role-Aware Handoff Memory (RAM), a source-grounded handoff interface that constructs bounded packets around downstream obligations. RAM extracts target-role obligations, links them to provenance-bearing evidence, prioritizes low-recoverability scientific details, checks packet-level sufficiency, and retrieves repair evidence for unsupported slots. Across execution- and retrieval-grounded handoff settings, RAM improves obligation-sensitive cases in ScienceAgentBench, CodeRAG-Bench, and a small code-only PaperBench subset, while producing mixed results on MLAgentBench tasks dominated by modeling, environment, or implementation-search bottlenecks. The results support a bounded conclusion: RAM is a targeted interface for preserving scientific obligations across role boundaries, not a universal replacement for full context, generic summarization, retrieval, or stronger downstream agents.

1 Introduction

Large language model (LLM) agents are increasingly used to automate role-decomposed scientific workflows, where different stages handle literature reading, hypothesis generation, experiment planning, code implementation, evaluation, and report writing (Lu et al., 2026; Schmidgall et al., 2025; Gottweis et al., 2025). This decomposition improves modularity, but it also creates a fragile handoff problem. At each role boundary, the downstream agent must act on information selected, compressed, or reformatted upstream, where small omissions can invalidate the final artifact (Cemri et al., 2025; Tang et al., 2025b).

The key failure is not simply missing context. A downstream agent may receive context that is locally plausible but insufficient for producing an evaluable scientific artifact (Huang et al., 2024; Starace et al., 2025). For example, when a planner hands work to a coder, the coder must preserve not only the high-level goal, but also the constraints that make the implementation valid under the evaluator: the data split, metric definition and direction, baseline protocol, API contract, output schema, file path, and evaluator-visible artifacts (Li et al., 2023b; Yang et al., 2024). A generic summary may mention the correct metric while omitting how it is computed, where the output must be saved, or which files the evaluator imports. The resulting code may run successfully while silently violating the protocol constraints that determine scientific and evaluative validity (Peters and Chin-Yee, 2025; Zhou et al., 2024).

We call these validity-critical requirements *downstream obligations*: action-conditioned requirements that the receiving role must satisfy under the task, paper, repository, rubric, or evaluator. Many obligations are *low-recoverability* details: easy to drop during summarization and difficult to reconstruct from general context (Padmakumar et al., 2025). We call this failure *handoff-induced obligation loss*, where requirements present in upstream evidence are weakened, omitted, or detached from provenance before downstream execution. Importantly, the same detail may be optional for one role but binding for another, making obligation loss role- and action-conditioned rather than ordinary relevance loss.

Existing context-management methods do not

directly target this interface. Long-context models expose more source material but still leave the receiving role to identify which details are binding constraints (Liu et al., 2024a). Summarization compresses prior context but may remove details that later become decisive for execution (Kang et al., 2025). Retrieval-augmented generation surfaces relevant evidence (Lewis et al., 2020), but relevance alone does not specify whether the next artifact satisfies metric, schema, API, file, or rubric obligations. Agent memory systems store and retrieve past state, but typically do not represent whether the retrieved state is sufficient for a specific downstream role, action, and evaluator (Packer et al., 2024; Xu et al., 2025). Scientific handoff therefore requires an interface that asks not only what prior information is relevant, but what the next artifact must satisfy.

We formalize this setting as *research handoff*. Given an upstream research state, a source role, a target role, a visible downstream task surface, and a packet budget, the goal is to construct a bounded handoff packet for a fixed downstream agent. The downstream model, action, evaluator, and execution budget are held fixed; only the transferred packet varies. This formulation separates handoff quality from general agent capability and allows us to test whether preserving downstream obligations improves scientific execution.

We propose ROLE-AWARE HANDOFF MEMORY (RAM), a source-grounded handoff interface for constructing obligation-preserving scientific packets. RAM represents a handoff not as a generic summary, but as typed target-role obligations linked to provenance-bearing evidence, recoverability labels, and packet-level sufficiency status. Given papers, task specifications, repository state, rubrics, and upstream traces, RAM extracts obligations for the receiving role, retrieves supporting evidence, prioritizes low-recoverability details, checks packet-level sufficiency, retrieves repair evidence for unsupported slots, and compiles a bounded packet that places artifact constraints before longer evidence passages. Across execution- and retrieval-grounded settings, RAM improves handoff-sensitive cases while showing mixed results when performance is dominated by modeling, environment, or broader implementation-search bottlenecks.

Our contributions are as follows:

- We identify *handoff-induced obligation loss* as a failure mode in role-decomposed scientific agents and formalize *research handoff* as a controlled packet-construction problem where the downstream agent, action, evaluator, and budget are fixed while only the transferred information varies.
- We introduce RAM, a role-aware handoff interface that builds bounded packets from typed downstream obligations, provenance-linked evidence, recoverability-aware prioritization, packet-level sufficiency checks, and targeted evidence repair.
- We evaluate RAM across execution- and retrieval-grounded handoff settings, showing gains in obligation-sensitive cases and mixed results when downstream performance is dominated by modeling, environment, or broader implementation-search bottlenecks.

2 Related Work

Automated Research Agents and Execution-Grounded Evaluation. LLM-based research agents have progressed from assisting individual scientific tasks, such as literature-grounded ideation, hypothesis generation, and draft construction (Wang et al., 2019; Qi et al., 2023), toward systems that coordinate broader research workflows. Recent systems increasingly decompose scientific work into staged pipelines for reading, planning, implementation, experimentation, analysis, and writing (Lu et al., 2026; Schmidgall et al., 2025; Gottweis et al., 2025; Tang et al., 2025a), and have been applied to wet-lab discovery, paper-centered code generation, and experiment reproduction (Swanson et al., 2025; Seo et al., 2026; Zhao et al., 2026). In parallel, execution-grounded benchmarks such as MLAgentBench, MLE-bench, ScienceAgentBench, PaperBench, and EXP-Bench evaluate whether agents produce valid code, experiments, and reproduction artifacts rather than only fluent text (Huang et al., 2024; Chan et al., 2025; Chen et al., 2025; Starace et al., 2025; Kon et al., 2025). These systems and benchmarks make role boundaries increasingly important: artifact failures may reflect not only weak downstream reasoning, but also missing constraints in the information passed between stages. Our work studies this transfer itself, asking whether the receiving role obtains the obligations needed for the next scientific action or artifact.

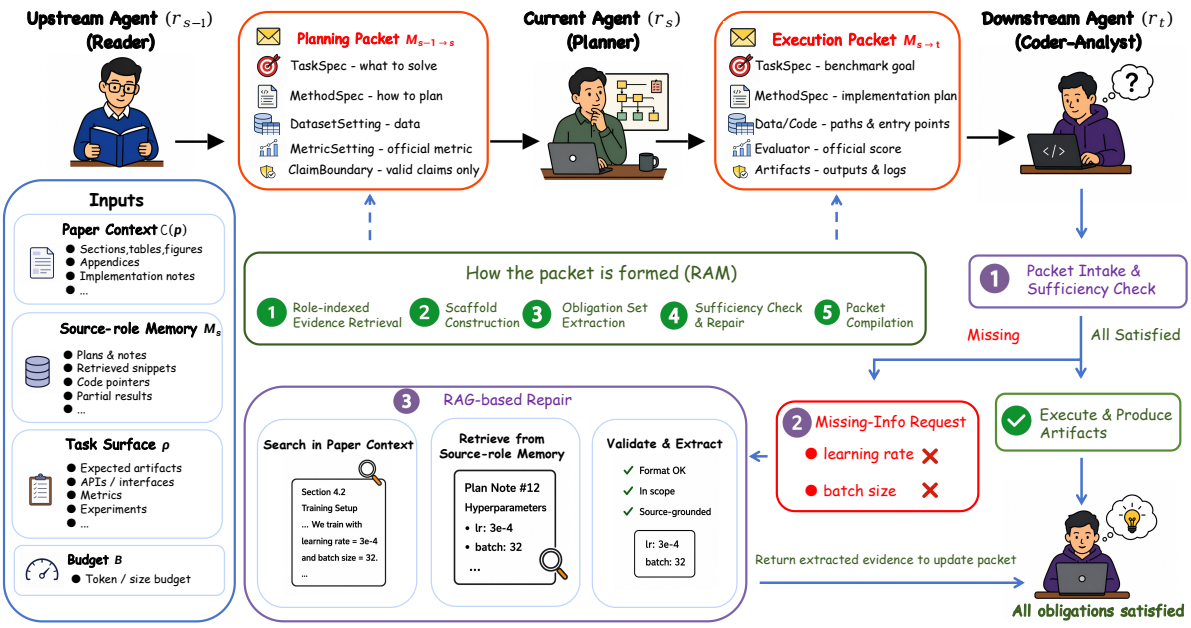

Figure 1: Overview of ROLE-AWARE HANDOFF MEMORY (RAM). Given upstream research state, a visible task surface, and a target downstream role, RAM constructs a bounded handoff packet organized around downstream obligations: requirements that the next action or artifact must satisfy to be executable, source-grounded, and evaluator-compatible. RAM retrieves provenance-bearing evidence, prioritizes low-recoverability details, checks packet-level sufficiency, retrieves repair evidence for unsupported slots when possible, and compiles a role-specific handoff packet.

Agent Memory, Context Compression, and Communication. A large body of work improves how agents retain, compress, retrieve, or exchange context. Role-based multi-agent frameworks organize collaboration through dialogue, role assignment, and workflow protocols (Wu et al., 2024; Li et al., 2023a; Hong et al., 2024; Qian et al., 2024), while failure analyses show that multi-agent systems often break when information is missing, misaligned, or not requested across agent boundaries (Cemri et al., 2025; Lin et al., 2026). Long-context models expose more source material, but do not guarantee that binding details are used (Liu et al., 2024a). Compression methods reduce context length, but can remove details that become decisive for execution (Kang et al., 2025). Memory and retrieval systems improve access to prior state, but typically optimize relevance or retention rather than whether the retrieved state is sufficient for a specific downstream role, action, and evaluator (Packer et al., 2024; Xu et al., 2025). Recent communication and context engineering methods further improve information exchange through pruning, latent thought sharing, evolving playbooks, or working-memory editing (Zhang et al., 2025; Zheng et al., 2025; Zhang et al., 2026a,b). In contrast, RAM treats cross-role transfer as obligation preservation: the packet should not only be compact or relevant, but sufficient for the downstream obligations that the next action or artifact must satisfy.

Context Construction and Structured Handoff Interfaces. RAM is related to retrieval-augmented and context-construction methods because it changes what information is provided to the downstream agent at inference time (Jiang et al., 2023; Trivedi et al., 2023; Asai et al., 2023). However, retrieval in RAM is a subroutine within a handoff procedure rather than the object of study. The evaluated object is the bounded handoff packet: how evidence is selected, organized, checked, repaired, and compiled for a target role under a fixed downstream agent, evaluator, and budget. In a handoff setting, this differs from retrieval-only interfaces, where relevant passages may be returned without specifying the file paths, APIs, metric directions, output schemas, or evaluator-visible artifacts that the downstream role must satisfy (Zhang et al., 2023; Liu et al., 2024b). It also differs from using a fixed packet template alone: RAM derives packet

fields from downstream-verifiable obligations and links them to provenance-bearing evidence. These approaches are complementary; our focus is the handoff interface that converts retrieved or remembered information into a role-specific packet for downstream scientific action.

3 Method

We study handoff construction for role-decomposed scientific agents. An upstream role produces intermediate research state, and a downstream role must use this state under a limited context budget to produce the next action or artifact, such as a retrieval query, implementation, experiment, metric report, log, or analysis. We isolate the handoff interface: the downstream agent, action, evaluator, and execution budget are fixed, and only the packet passed across the role boundary is changed.

We formulate this problem as *budgeted obligation preservation*. A handoff packet should not merely summarize prior context or retrieve relevant passages; it should preserve downstream-verifiable requirements that the receiving role must satisfy. We call these requirements *obligations*. Examples include using the official data split, computing the correct metric, preserving metric direction, saving required output files, exposing repository APIs, and satisfying rubric-visible artifact surfaces.

ROLE-AWARE HANDOFF MEMORY (RAM) is a provenance-bearing packet construction procedure. It uses fixed obligation schemas, role/action metadata, recoverability-aware evidence ranking, packet-level sufficiency checks, and targeted evidence repair. RAM does not train a retriever, introduce learned benchmark-specific thresholds, modify the downstream agent, access hidden labels or test feedback, or alter the evaluator.

3.1 Research Handoff Setting

Let p denote the target paper or task instance, r_s the source role, r_t the target role, and a_t the downstream action. RAM constructs a bounded handoff packet from the available context $\mathcal{C}(p)$, source-role memory $\mathcal{M}_s$, visible downstream task surface ρ, and packet budget B:

$$m = H(\mathcal{C}(p), \mathcal{M}_s, r_s, r_t, a_t, \rho, B), \qquad |m| \leq B. \tag{1}$$

Here ρ contains only materials available before downstream execution. The packet is given to a fixed downstream agent A, which produces an output $y = A(m)$, such as a retrieval query, code artifact, experiment output, or reproduction artifact. The output is evaluated by a fixed benchmark-specific scoring function $S(y)$. Within each setting, the only intervention is how m is constructed.

3.2 Obligations and Evidence Units

RAM converts the visible task surface into typed downstream obligations:

$$\mathcal{O}(\rho, r_t, a_t) = \{o_1, \ldots, o_k\}. \tag{2}$$

Each obligation corresponds to a requirement that can affect execution or evaluation for the target role and action. Obligation types include method implementation, dataset setup, metric computation, baseline comparison, experiment configuration, artifact creation, evaluator API preservation, and official submission generation. The schemas are fixed before evaluation; benchmark adapters map visible instance materials to these schemas without using hidden labels or benchmark outcomes.

Each obligation o_i has a type $\tau(o_i)$ and a set of required slots $\mathrm{Slots}(o_i)$. Slots specify the information needed for the obligation to be actionable. For example, a metric obligation may require the metric name, direction, computation rule, and reporting location; an artifact obligation may require the output path, format, schema, and evaluator dependency.

RAM also converts context and source-role memory into typed evidence units:

$$u = (x, t, \pi, \mathcal{R}, \mathcal{A}, \gamma, \sigma), \tag{3}$$

where x is the content, t is the unit type, π is provenance, $\mathcal{R}$ is the set of relevant roles, $\mathcal{A}$ is the set of supported downstream actions, γ is recoverability, and σ is the set of obligation slots supported by the unit. Provenance may point to a paper section, repository file, task instruction, visible evaluator script, rubric item, or source-role memory entry. Supported slots σ are assigned using fixed slot-matching templates; they indicate whether the packet contains support for a slot, not whether the final artifact will be semantically correct.

Recoverability is fixed by unit type. Low-recoverability units include details that a downstream agent is unlikely to reconstruct reliably after handoff, such as exact metric definitions, data splits, hyperparameters, baseline protocols, repository entry points, file paths, and output schemas. High-level goals and general background are treated as

more recoverable. The full schema, unit types, recoverability labels, and slot-matching templates are provided in Appendix B.

3.3 Evidence Selection and Targeted Repair

For each obligation o, RAM creates a fixed query plan $Q(o)$ from the obligation type and required slots. Candidate chunks are retrieved from $\mathcal{C}(p)$ and $\mathcal{M}_s$ using lexical matches and implementation-relevant section headings. RAM then ranks candidate evidence for the target edge $e = (r_t, a_t)$ with a fixed additive score combining lexical relevance, role match, action support, required unit type, recoverability, and provenance.

After evidence assignment, RAM checks packet-level sufficiency. For each obligation o_i, let U_i denote its assigned evidence units:

$$g(o_i, U_i) = \begin{cases} \text{satisfied}, & \text{Slots}(o_i) \subseteq \bigcup_{u \in U_i} \sigma(u), \\ \text{missing}, & \text{otherwise}. \end{cases} \tag{4}$$

The checker verifies whether the packet contains support for required slots; it does not verify final semantic correctness or guarantee that the generated artifact will pass evaluation. For example, naming a metric is insufficient if the packet omits its direction, computation rule, or reporting location.

When an obligation is missing, RAM issues a deterministic missing-slot repair query:

$$q_i = \phi(o_i, r_t, a_t, \text{MissingSlots}(o_i, U_i)). \tag{5}$$

The query asks for compact, source-grounded evidence for the missing slots and uses the same retrieval heuristic. Repair first searches $\mathcal{M}_s$ and then falls back to $\mathcal{C}(p)$. If evidence is found, the repaired unit is added and the checker is rerun. If no direct evidence is found within budget, the obligation remains unresolved in the packet. Thus, repair improves packet support for unsupported slots; it does not repair the downstream artifact itself.

3.4 Packet Compilation

RAM compiles selected units into a fixed-order handoff packet. The packet begins with a *contract block* stating the source role, target role, downstream action, and benchmark target. It then lists *artifact surfaces*, including evaluator-facing entry points, APIs, output schemas, result files, logs, and directories. The *obligation block* states downstream requirements and required slots, followed by a *sufficiency block* that records supported and missing slots. A *repair block* adds retrieved support for missing slots or marks unresolved obligations. The packet ends with trimmed evidence units, including their type, recoverability, provenance, supported slots, and content.

This order is an execution-priority heuristic. Executable constraints, artifact surfaces, and obligation-level requirements are placed before longer evidence passages so that the packet first exposes non-negotiable downstream requirements. If the packet exceeds budget, lower-priority evidence is trimmed first and an explicit truncation marker is inserted.

3.5 Task Surface Construction and Scope

RAM uses a shared packet schema, while each benchmark exposes its visible task surface ρ from materials available in that setting, such as paper text, repository files, visible evaluation scripts, required outputs, rubric items, and source-role workflow notes. Official artifact constraints may be included only when visible in the instance materials; they are not treated as discoveries made by RAM.

For CodeRAG-Bench, RAM is used as a retrieval-facing instantiation of the same handoff principle. Instead of compiling a packet for downstream generation, it compiles role, action, unit-type, provenance, and search-term fields into a query representation for the fixed retrieval backend. This keeps the retrieval backend and evaluation protocol unchanged while testing whether obligation-aware handoff improves the query interface.

These construction rules do not change the downstream evaluator, access hidden labels, use hidden test feedback, or introduce learned benchmark-specific thresholds. Thus, RAM is evaluated as a handoff packet construction method, not as a standalone retriever, a new downstream agent, or a replacement evaluator.

4 Experiments

4.1 Experimental Setup

Evaluation protocol. We evaluate RAM as a handoff intervention rather than as a new downstream agent. In each setting, the downstream action, benchmark instance, evaluator, retrieval backend when applicable, and execution budget are fixed, isolating the handoff packet as the only intervention. RAM uses only task materials available

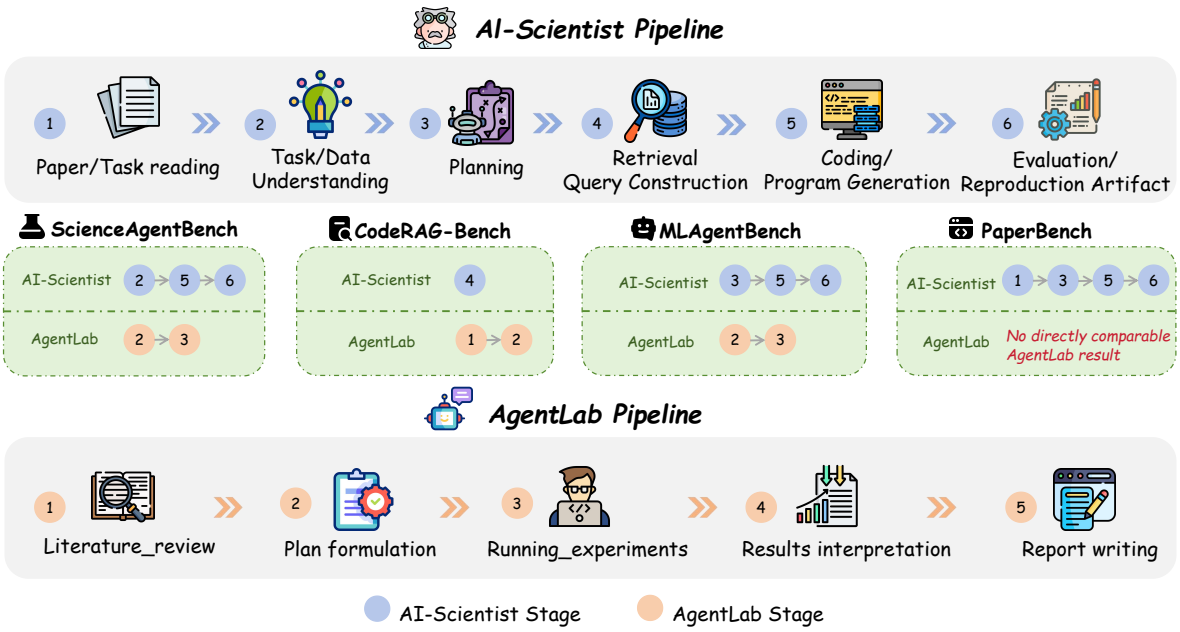

Figure 2: Benchmark coverage across research-agent handoff stages. Numbers denote stages in the corresponding AI-Scientist or AgentLab pipeline. ScienceAgentBench and MLAgentBench test execution-facing handoff, CodeRAG-Bench tests retrieval-query handoff, and PaperBench reproduction tests paper-to-code handoff. In all settings, only the handoff packet is varied.

Paper	Base	RAM
AP	23.4	**26.8**
MU	62.2	**74.3**
RC	30.9	**32.3**
SNSE	39.2	**51.2**
SoT-CFG	48.2	**51.9**
SI	57.3	**67.0**

Table 1: PaperBench code-only reproduction results on the controlled six-paper subset selected by PaperBench rubric size. *Note.* AP = adaptive-pruning; MU = mechanistic-understanding; RC = robust-clip; SNSE = sequential-neural-score-estimation; SoT-CFG = stay-on-topic-with-classifier-free-guidance; SI = stochastic-interpolants.

before downstream execution and does not access hidden labels, hidden tests, or post-hoc evaluator feedback.

Benchmarks. We evaluate four handoff settings. **ScienceAgentBench** (Chen et al., 2025) evaluates scientific program generation, where the packet must preserve task instructions, dataset information, required outputs, and evaluator constraints. **CodeRAG-Bench** (Wang et al., 2025) evaluates retrieval-facing handoff, where upstream task state is converted into implementation-relevant retrieval queries. **MLAgentBench** (Huang et al., 2024) evaluates ML artifact construction, where packets must preserve file targets, output schemas, evaluator APIs, and metric directions. **PaperBench** (Starace et al., 2025) evaluates paper-to-code handoff; we report code-only results on a controlled six-paper subset selected by PaperBench rubric size, using papers with fewer than 200 rubric nodes, rather than a full PaperBench leaderboard result. Figure 2 summarizes how these benchmarks instantiate different handoff points in the AI-Scientist (Lu et al., 2026) and AgentLab (Schmidgall et al., 2025) pipelines.

Baselines and controls. Because RAM is a packet-construction method, the main comparison is against the corresponding base handoff interface in each system. When applicable, we include diagnostic controls such as generic summaries, retrieval-oriented handoff, and ablations that remove role, provenance, or checker information. We do not benchmark standalone RAG algorithms; retrieval is treated as a subroutine inside a fixed handoff interface.

Metrics. We report benchmark-native metrics. ScienceAgentBench uses Success Rate (SR), CodeBERTScore (CBS), and Valid Execution Rate (VER). CodeRAG-Bench uses NDCG, Recall, and MRR from the official BEIR retrieval evaluation.

System	Method	SR	CBS	VER
AgentLab	Base	8.8	74.7	31.4
	RAM	**10.8**	**76.2**	**38.2**
AI-Scientist	Base	19.6	72.8	39.2
	RAM	**21.6**	**74.3**	**43.1**

Table 2: ScienceAgentBench results under AgentLab and AI-Scientist. RAM is compared with the corresponding base handoff under the same downstream action and evaluator.

MLAgentBench uses task-specific official scores, so we report per-task scores without averaging heterogeneous metrics. PaperBench uses code-only rubric scores on the six-paper subset. Unless otherwise stated, results are fixed-budget execution runs rather than significance-tested leaderboard submissions.

4.2 Main Results

Scientific execution. Table 2 reports ScienceAgentBench results under AgentLab and AI-Scientist. RAM improves SR, CBS, and VER in both systems. The absolute SR gains are modest, but the improvements are consistent across the two systems and all three metrics, suggesting that structured handoff helps when execution depends on preserving task instructions, dataset constraints, required outputs, and evaluator-visible checks.

Retrieval-facing handoff. Table 3 reports CodeRAG-Bench macro-average results over MBPP and ODEX. Both RAM variants improve over the BM25 original-query baseline across Top-10 and Top-100 metrics, with the largest gains in Recall. This suggests that role- and provenance-aware query construction helps surface more implementation-relevant evidence. We interpret this as retrieval-facing handoff rather than as a claim that RAM is a standalone retriever.

Paper-to-code reproduction. Table 1 reports PaperBench code-only reproduction on a node < 200 six-paper subset. RAM improves over the base handoff on all six papers, with the largest gains on mechanistic-understanding and sequential-neural-score-estimation. These cases require the downstream code artifact to preserve multiple method, experiment, and rubric-visible implementation constraints. Since this is a small controlled subset, we treat it as evidence for paper-to-code handoff behavior rather than as a full PaperBench leaderboard claim.

ML artifact construction. Table 4 reports MLAgentBench task-level official scores. Because MLAgentBench tasks use heterogeneous metrics, we do not average them across tasks. RAM improves several reported scored tasks under both systems, including Vectorization, Spaceship Titanic, AMP Parkinsons, and BabyLM. These tasks are consistent with the intended use case of RAM: preserving evaluator-facing artifacts, submission formats, output contracts, and metric directions.

The result is not uniform. RAM underperforms the base on CIFAR-10 under AI-Scientist and on Feedback under both systems. These mixed outcomes support a bounded interpretation: RAM can improve artifact-oriented handoff, but it is not a universal replacement for original context, task-specific modeling choices, or broader implementation search.

4.3 Ablation and Component Analysis

We analyze two complementary settings: execution-facing handoff in ScienceAgentBench and retrieval-facing handoff on the ODEX subset of CodeRAG-Bench.

Execution-facing components. Table 5 reports the ScienceAgentBench ablation under AgentLab. RAM obtains the best SR and CBS. Removing role, provenance, or checker information lowers SR below both RAM and the base pipeline, suggesting that gains come from coherent packet structure rather than from a single field or formatting alone.

The ablation also shows that valid execution is not equivalent to task success: some ablated variants retain relatively high VER but substantially lower SR. We therefore interpret these ablations as packet-coherence stress tests, not as proof that every field independently improves every task.

Retrieval-facing components. Table 6 reports an ODEX-only ablation. We isolate ODEX because the macro-average in Table 3 combines MBPP with a more challenging retrieval subset. Under both systems, RAM improves over the base query, while removing provenance weakens the result. Role and checker fields have smaller and less consistent effects, which is expected in a retrieval-only setting where provenance directly affects query grounding.

4.4 Boundary and Failure Analysis

The results indicate that RAM is most useful when the bottleneck is the handoff interface itself.

Method	Top-10			Top-100		
	NDCG@10 ↑	Recall@10 ↑	MRR@10 ↑	NDCG@100 ↑	Recall@100 ↑	MRR@100 ↑
BM25	46.5	52.7	44.6	49.0	64.6	45.1
RAM (AI-Scientist)	48.0 (+1.5)	56.8 (+4.0)	45.3 (+0.6)	50.9 (+1.8)	70.6 (+5.9)	45.9 (+0.7)
RAM (AgentLab)	48.9 (+2.5)	58.0 (+5.3)	46.2 (+1.5)	51.2 (+2.1)	68.9 (+4.2)	46.6 (+1.4)

Table 3: CodeRAG-Bench macro-average results over MBPP and ODEX using official BEIR metrics. Parenthesized values show absolute improvement over the BM25 original-query baseline.

Task	Official Metric	AI-Scientist		AgentLab	
		Base Score	RAM Score	Base Score	RAM Score
Vectorization	Wall-clock time (s) ↓	3.396	**3.190**	3.191	**3.121**
Spaceship Titanic	Accuracy ↑	0.7228	**0.7648**	0.4761	**0.7211**
CIFAR-10	Accuracy ↑	**0.7920**	0.5980	-	-
Feedback	MCRMSE ↓	**0.593**	0.617	**0.583**	0.631
AMP Parkinsons	SMAPE+1 ↓	63.6	**58.7**	167.6	**66.1**
BabyLM	Perplexity ↓	2650.7	**2504.2**	2579.6	**2499.2**

Table 4: MLAgentBench per-task official metrics. Dashes indicate that no directly comparable run was available under that system.

Method	**SR**	**CBS**	**VER**
Base	8.8	74.7	31.4
RAM	**10.8**	**76.2**	**38.2**
RAM w/o Role	6.9	73.9	37.3
RAM w/o Provenance	6.9	73.9	33.3
RAM w/o Checker	6.9	75.5	32.4

Table 5: ScienceAgentBench component ablation under AgentLab.

System	**Method**	**N@10 ↑**	**R@10 ↑**	**M@10 ↑**
AI-Sci.	Base	6.8	11.7	5.7
	RAM	**12.1**	**18.2**	**10.5**
	w/o Prov.	10.2	16.2	8.6
AgentLab	Base	6.8	11.7	5.7
	RAM	**10.8**	**19.2**	8.6
	w/o Role	10.7	17.7	**9.0**
	w/o Prov.	10.0	17.2	8.2
	w/o Check.	**10.8**	18.9	8.7

Table 6: CodeRAG-Bench ODEX ablation. The main CodeRAG-Bench table reports the macro-average over MBPP and ODEX. *Note.* N@10, R@10, and M@10 denote NDCG@10, Recall@10, and MRR@10.

ScienceAgentBench and PaperBench require the downstream stage to preserve task constraints, artifact surfaces, metrics, and source-grounded implementation details. CodeRAG-Bench shows that the same handoff structure can also help before generation by improving the query used to retrieve implementation-relevant evidence.

MLAgentBench identifies the main boundary. RAM helps several artifact-oriented tasks but does not dominate across settings. CIFAR-10 under AI-Scientist appears more sensitive to visual-data preprocessing, model architecture, augmentation, and training heuristics that are only partially represented in the handoff packet. Feedback is another negative case under both systems. These failures suggest that RAM is less effective when success depends on domain-specific modeling, environment assumptions, or broader implementation search rather than recovering evaluator-facing obligations. RAM should therefore be understood as a targeted handoff interface, not a universal replacement for original context, summarization, retrieval, or stronger downstream agents.

5 Conclusion

We presented RAM, a role-aware handoff interface that treats stage-to-stage transfer in automated research agents as budgeted obligation preservation. RAM constructs bounded packets around downstream obligations, provenance-bearing evidence, packet-level sufficiency checks, and targeted evidence repair, while leaving the downstream agent and evaluator unchanged.

Results across 4 benchmarks show that RAM helps when downstream success depends on preserving low-recoverability scientific details such as metrics, artifact schemas, API contracts, and file targets. The mixed results also clarify its scope: RAM is a targeted handoff interface, not a universal replacement for full context, retrieval, summarization, or stronger agents.

Limitations

This work evaluates RAM as a handoff interface rather than as a new downstream agent or evaluator. The conclusions are therefore bounded by the host architectures, benchmark adapters, and execution budgets used in our experiments.

First, RAM does not solve failures caused by missing dependencies, inaccessible data, hidden evaluator contracts, environment incompatibilities, or runtime errors. Second, structured packets can under-select or over-constrain context when a task requires broad paper-level understanding, exploratory reasoning, or domain-specific implementation search. Third, MLAgentBench uses heterogeneous task metrics, so its results should be interpreted at the task level rather than as a single aggregate score. Fourth, our PaperBench evaluation uses a small controlled subset rather than a full benchmark leaderboard evaluation. Finally, RAM currently uses fixed handoff policies; future work should study adaptive selection among original context, summarization, retrieval-only handoff, RAM, and hybrid context access.

References

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *Preprint*, arXiv:2310.11511.

Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. 2025. Why do multi-agent llm systems fail? *Preprint*, arXiv:2503.13657.

Jun Shern Chan, Neil Chowdhury, Oliver Jaffe, James Aung, Dane Sherburn, Evan Mays, Giulio Starace, Kevin Liu, Leon Maksin, Tejal Patwardhan, Lilian Weng, and Aleksander Mądry. 2025. MLE-bench: Evaluating machine learning agents on machine learning engineering. *Preprint*, arXiv:2410.07095.

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. 2025. ScienceAgentBench: Toward rigorous assessment of language agents for data-driven scientific discovery. In *Proceedings of the Thirteenth International Conference on Learning Representations*.

Juraj Gottweis, Wei-Hung Weng, Alexander Daryin, Tao Tu, Anil Palepu, Petar Sirkovic, Artiom Myaskovsky, Felix Weissenberger, Keran Rong, Ryutaro Tanno, Khaled Saab, Dan Popovici, Jacob Blum, Fan Zhang, Katherine Chou, Avinatan Hassidim, Burak Gokturk, Amin Vahdat, Pushmeet Kohli, and 15 others. 2025. Towards an AI co-scientist. *Preprint*, arXiv:2502.18864.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *Proceedings of the Twelfth International Conference on Learning Representations*.

Qian Huang, Jian Vora, Percy Liang, and Jure Leskovec. 2024. MLAgentBench: Evaluating language agents on machine learning experimentation. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 20271–20309. PMLR.

Zhengbao Jiang, Frank Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active retrieval augmented generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7969–7992. Association for Computational Linguistics.

Minki Kang, Wei-Ning Chen, Dongge Han, Huseyin A. Inan, Lukas Wutschitz, Yanzhi Chen, Robert Sim, and Saravan Rajmohan. 2025. ACON: Optimizing context compression for long-horizon LLM agents. *Preprint*, arXiv:2510.00615.

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. 2025. EXP-bench: Can AI conduct AI research experiments? *Preprint*, arXiv:2505.24785.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. 2023a. CAMEL: Communicative agents for "mind" exploration of large language model society. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. 2023b. API-bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods*

in Natural Language Processing, pages 3102–3116. Association for Computational Linguistics.

Bohan Lin, Kuo Yang, Zelin Tan, Yingchuan Lai, Chen Zhang, Guibin Zhang, Xinlei Yu, Miao Yu, Xu Wang, Yudong Zhang, and Yang Wang. 2026. AgentAsk: Multi-agent systems need to ask. *Preprint*, arXiv:2510.07593.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Tianyang Liu, Canwen Xu, and Julian McAuley. 2024b. Repobench: Benchmarking repository-level code auto-completion systems. In *International Conference on Learning Representations*, volume 2024, pages 47832–47850.

Chris Lu, Cong Lu, Robert Tjarko Lange, Yutaro Yamada, Shengran Hu, Jakob Foerster, David Ha, and Jeff Clune. 2026. Towards end-to-end automation of AI research. *Nature*, 651(8107):914–919.

Charles Packer, Sarah Wooders, Kevin Lin, Vivian Fang, Shishir G. Patil, Ion Stoica, and Joseph E. Gonzalez. 2024. MemGPT: Towards LLMs as operating systems. *Preprint*, arXiv:2310.08560.

Vishakh Padmakumar, Zichao Wang, David Arbour, and Jennifer Healey. 2025. Principled content selection to generate diverse and personalized multi-document summaries. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 29884–29899. Association for Computational Linguistics.

Uwe H. Peters and Benjamin Chin-Yee. 2025. Generalization bias in large language model summarization of scientific research. *Royal Society Open Science*, 12(4):241776.

Biqing Qi, Kaiyan Zhang, Haoxiang Li, Kai Tian, Sihang Zeng, Zhang-Ren Chen, and Bowen Zhou. 2023. Large language models are zero shot hypothesis proposers. *Preprint*, arXiv:2311.05965.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186. Association for Computational Linguistics.

Samuel Schmidgall, Yusheng Su, Ze Wang, Ximeng Sun, Jialian Wu, Xiaodong Yu, Jiang Liu, Michael Moor, Zicheng Liu, and Emad Barsoum. 2025. Agent laboratory: Using LLM agents as research assistants. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 5977–6043. Association for Computational Linguistics.

Minju Seo, Jinheon Baek, Seongyun Lee, and Sung Ju Hwang. 2026. Paper2Code: Automating code generation from scientific papers in machine learning. In *Proceedings of the Fourteenth International Conference on Learning Representations*.

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. 2025. PaperBench: Evaluating AI's ability to replicate AI research. In *Proceedings of the Forty-Second International Conference on Machine Learning*.

Kyle Swanson, Wesley Wu, Nash L. Bulaong, John E. Pak, and James Zou. 2025. The virtual lab of AI agents designs new SARS-CoV-2 nanobodies. *Nature*, 646(8085):716–723.

Jiabin Tang, Lianghao Xia, Zhonghang Li, and Chao Huang. 2025a. Ai-researcher: Autonomous scientific innovation. In *Advances in Neural Information Processing Systems*, volume 38, pages 9481–9520. Curran Associates, Inc.

Yichen Tang, Weihang Su, Yujia Zhou, Yiqun Liu, Min Zhang, Shaoping Ma, and Qingyao Ai. 2025b. Augmenting multi-agent communication with state delta trajectory. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 10219–10240. Association for Computational Linguistics.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2023. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10014–10037. Association for Computational Linguistics.

Qingyun Wang, Lifu Huang, Zhiying Jiang, Kevin Knight, Heng Ji, Mohit Bansal, and Yi Luan. 2019. PaperRobot: Incremental draft generation of scientific ideas. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1980–1991. Association for Computational Linguistics.

Zora Zhiruo Wang, Akari Asai, Xinyan Velocity Yu, Frank F. Xu, Yiqing Xie, Graham Neubig, and Daniel Fried. 2025. CodeRAG-bench: Can retrieval augment code generation? In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 3199–3214. Association for Computational Linguistics.

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. 2024. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *Proceedings of the First Conference on Language Modeling*.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. 2025. A-MEM: Agentic memory for LLM agents. *Preprint*, arXiv:2502.12110.

John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik R Narasimhan, and Ofir Press. 2024. SWE-agent: Agent-computer interfaces enable automated software engineering. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Fengji Zhang, Bei Chen, Yue Zhang, Jacky Keung, Jin Liu, Daoguang Zan, Yi Mao, Jian-Guang Lou, and Weizhu Chen. 2023. RepoCoder: Repository-level code completion through iterative retrieval and generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2471–2484. Association for Computational Linguistics.

Guibin Zhang, Yanwei Yue, Zhixun Li, Sukwon Yun, Guancheng Wan, Kun Wang, Dawei Cheng, Jeffrey Xu Yu, and Tianlong Chen. 2025. Cut the crap: An economical communication pipeline for LLM-based multi-agent systems. In *Proceedings of the Thirteenth International Conference on Learning Representations*.

Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, Urmish Thakker, James Zou, and Kunle Olukotun. 2026a. Agentic context engineering: Evolving contexts for self-improving language models. *Preprint*, arXiv:2510.04618.

Yuxiang Zhang, Jiangming Shu, Ye Ma, Xueyuan Lin, Shangxi Wu, and Jitao Sang. 2026b. Memory as action: Autonomous context curation for long-horizon agentic tasks. *Preprint*, arXiv:2510.12635.

Xuanle Zhao, Zilin Sang, Yuxuan Li, Qi Shi, Weilun Zhao, Shuo Wang, Duzhen Zhang, Xu Han, Zhiyuan Liu, and Maosong Sun. 2026. AutoReproduce: Automatic AI experiment reproduction with paper lineage. *Preprint*, arXiv:2505.20662.

Yujia Zheng, Zhuokai Zhao, Zijian Li, Yaqi Xie, Mingze Gao, Lizhu Zhang, and Kun Zhang. 2025. Thought communication in multiagent collaboration. In *Advances in Neural Information Processing Systems*, volume 38, pages 123389–123418. Curran Associates, Inc.

Lexin Zhou, Wout Schellaert, Fernando Martínez-Plumed, Yael Moros-Daval, Cèsar Ferri, and José Hernández-Orallo. 2024. Larger and more instructable language models become less reliable. *Nature*, 634:61–68.

A Appendix Roadmap

Scope. This appendix provides implementation and evaluation details that support the main paper without changing its claims. We include RAM schemas, packet construction details, a mechanism case study, CodeRAG-Bench breakdowns, MLAgentBench evaluation details, PaperBench subset results, and failure analyses. All packets are constructed from materials available before downstream execution; hidden labels, hidden tests, and post-hoc evaluator feedback are not used.

Section B describes the fixed obligation and evidence schemas and summarizes packet blocks and trimming. Section C gives a mechanism case study comparing the base handoff with RAM. Section D reports CodeRAG-Bench per-dataset results. Section E provides MLAgentBench evaluation details. Section F reports the PaperBench six-paper subset. Section G summarizes failure modes.

B RAM Obligation and Evidence Schemas

RAM uses fixed schemas to construct obligations, evidence units, recoverability labels, and slot-level support indicators. The schemas are defined before evaluation and are not tuned using hidden labels, private test feedback, or benchmark outcomes. Benchmark adapters only map visible instance materials into these shared categories.

B.1 Obligation Types

See Table 7.

B.2 Evidence Recoverability and Slot Support

See Table 8.

B.3 Benchmark Adapters and Schema Scope

Each benchmark exposes a different visible task surface ρ. RAM uses a fixed adapter for each benchmark to map visible materials into the shared obligation and evidence schemas. The adapters do not access hidden labels, private tests, or post-hoc evaluator feedback.

ScienceAgentBench. The task surface includes task instructions, dataset descriptions, required program outputs, visible evaluation constraints, and source-role notes when available.

CodeRAG-Bench. The task surface is converted into a retrieval-facing representation. RAM maps role, action, unit type, provenance, and search terms into queries for the same retrieval backend used by the baseline.

MLAgentBench. The task surface includes official task folders, visible training and evaluation interfaces, expected output formats, repository constraints, and task-specific metrics.

PaperBench. The task surface includes paper text, implementation details, repository scaffolds when available, and rubric-visible code artifact targets.

Schema boundary. The schemas identify whether a handoff packet contains source-grounded support for downstream obligations. They do not verify final semantic correctness, replace the benchmark evaluator, or guarantee successful execution. If a required detail is absent from all visible materials, RAM marks the corresponding slot as unresolved rather than inferring it from hidden information.

Obligation type	Required slots	Visible evidence examples
Method implementation	Method name, components, algorithmic steps, inputs, outputs.	Method section, algorithm box, README, implementation notes.
Dataset setup	Dataset name, split, preprocessing, input format, data path or loader.	Task instructions, data description, repository loader, benchmark layout.
Metric computation	Metric name, direction, computation rule, aggregation, reporting location.	Evaluation section, visible evaluator script, metric description.
Baseline comparison	Baseline name, protocol, configuration, comparison target, reporting format.	Experiment section, baseline table, reproduction instruction, rubric item.
Experiment configuration	Hyperparameters, training budget, model configuration, seeds if specified.	Appendix, implementation details, config files, command-line scripts.
Artifact creation	File path, format, output schema, directory, logs, checkpoints.	Task instructions, visible evaluator interface, sample submission, scaffold.
Evaluator API preservation	Entry point, function signature, import path, return type, dependency surface.	Visible evaluation script, unit-test interface, repository API documentation.
Official submission generation	Submission name, column or field schema, ordering, packaging rule.	Sample submission, benchmark instruction, visible scoring script.

Table 7: Obligation types and required slots used by RAM. These slots define packet-level support requirements; they do not guarantee final artifact correctness or high scores.

Evidence type	Recoverability	Rationale
Metric definition	Low	Exact metric, direction, aggregation, and reporting location are easy to omit and hard to infer.
Data split or data path	Low	Official split, directory layout, or data loader often must match the evaluator.
Output schema	Low	Columns, fields, filenames, and directory structures are evaluator-facing and brittle.
Repository API or entry point	Low	Function names, import paths, signatures, and return types must match evaluator expectations.
Hyperparameter or configuration	Low	Training budget, model size, learning rate, batch size, and preprocessing flags are often scattered.
Baseline protocol	Low	Baseline settings and comparison protocols are difficult to reconstruct without source evidence.
Rubric-visible implementation target	Low	Rubric leaves or reproduction targets specify artifact surfaces that must be implemented.
Algorithmic step	Medium	Core method steps are important but may sometimes be inferred from surrounding text.
High-level task goal	Higher	The overall objective is usually repeated in task instructions, titles, or summaries.
General motivation	Higher	Background context is useful but less likely to determine evaluator compatibility.

Table 8: Type-level recoverability labels used by RAM. Low-recoverability units are prioritized because their omission is more likely to cause obligation loss.

Slot	Support criterion	Insufficient support
Metric name	The evidence names the metric to be optimized or reported.	The evidence only says "evaluate performance".
Metric direction	The evidence states whether higher or lower is better, or this is explicit in a visible evaluator.	The metric is named but direction is omitted.
Computation rule	The evidence specifies the formula, aggregation, or visible evaluator call.	The metric is mentioned without computation details.
Output path	The evidence gives the required file, directory, or save location.	The evidence only says to save results.
Output schema	The evidence specifies columns, fields, shape, type, or format.	A file name is given without the expected schema.
API entry point	The evidence specifies the function, script, import path, or callable interface.	The task is described without the evaluator-facing interface.
Dataset split	The evidence identifies the official split, file names, or loader.	The dataset is named without split or loader details.
Hyperparameter	The evidence provides a value, range, or configuration source.	The evidence only says to tune or train a model.
Rubric target	The evidence identifies a concrete rubric-visible artifact or behavior.	The evidence describes the paper goal without a benchmark-visible target.

Table 9: Slot-level support criteria used for assigning $\sigma(u)$. The checker requires packet support for slots, but does not certify downstream implementation.

Packet block	Function	Trimming priority
Contract	States source role, target role, downstream action, and benchmark target.	Protected.
Artifact surface	Lists evaluator-facing files, APIs, output schemas, logs, directories, and entry points.	Protected.
Obligations	Lists downstream requirements and required slots.	Protected.
Sufficiency status	Marks supported obligations and missing slots.	Protected.
Repair evidence	Adds compact evidence retrieved for missing slots or marks unresolved slots.	Trim only after selected evidence.
Selected evidence	Provides provenance-bearing evidence units with type, recoverability, slots, and compact content.	Trim first, starting from the lowest-ranked evidence.

Table 10: RAM packet blocks and trimming policy. Executable constraints and evaluator-facing surfaces are preserved before longer supporting evidence.

C Mechanism Case Study: Base Handoff vs. RAM

Purpose. This case study illustrates the mechanism of RAM. It is not an additional benchmark result or leaderboard claim. It shows how RAM differs from a base handoff interface by exposing obligations, checking packet support, and retrieving repair evidence for unsupported slots.

Consider a paper-to-code handoff where an upstream reader or planner passes a paper and reproduction objective to a downstream coder. The relevant information is distributed across the method, experiments, appendix, tables, and implementation notes. The risk is that the downstream coder receives a plausible context while missing low-recoverability details such as dataset splits, metric direction, baseline protocol, configuration values, artifact paths, or evaluator-visible outputs.

Base handoff. The base interface passes available context to the downstream stage without explicitly representing which requirements must be satisfied. The coder may see the high-level method but miss binding constraints. Missing details do not necessarily appear as explicit errors, so the downstream agent may silently choose defaults.

RAM handoff. RAM converts the same visible materials into typed obligations and evidence units. For the downstream coding action, it prioritizes low-recoverability details and checks whether selected evidence supports the required slots for each downstream obligation.

Illustrative repair trace. Suppose the initial packet supports method implementation, dataset setup, metric computation, and experiment configuration, but lacks baseline comparison and artifact creation. RAM marks these as missing slots and issues targeted queries for source-grounded evidence about the baseline protocol and required output artifacts. If evidence is found, it is added to the packet; otherwise, the slots remain explicit unresolved blockers.

The key difference is that RAM does not infer unsupported facts or repair the generated artifact directly. It changes the handoff state: missing requirements become named missing slots, targeted repair queries, or explicit unresolved blockers. This prevents the downstream agent from silently replacing unknown baselines, output schemas, or artifact paths with convenient defaults.

Aspect	Base handoff	RAM handoff
Information form	Compact context or task state.	Obligation-centered packet with provenance-bearing evidence.
Missing details	May remain implicit or unnoticed.	Become missing slots or unresolved blockers.
Artifact constraints	May be mixed with narrative context.	Placed early as evaluator-facing artifact surfaces.
Repair behavior	No explicit repair step.	Retrieves source-grounded evidence for unsupported slots.
Downstream risk	The agent may choose silent defaults.	The agent sees which obligations are supported or missing.

Table 11: Mechanism comparison between base handoff and RAM. The case study supports the method claim that RAM makes obligation loss explicit; it is not an additional benchmark result.

D Full CodeRAG-Bench Results

The main paper reports macro-average CodeRAG-Bench results over MBPP and ODEX. Here we provide the per-dataset breakdown for the controlled comparison between the base query interface and RAM. ODEX is the more difficult subset in our runs and is therefore also used for component analysis in the main paper.

System	Method	N@10	R@10	M@10
AI-Scientist	Base	6.8	11.7	5.7
AI-Scientist	RAM	**12.1**	**18.2**	**10.5**
AgentLab	Base	6.8	11.7	5.7
AgentLab	RAM	**10.8**	**19.2**	**8.6**

Table 12: CodeRAG-Bench ODEX results comparing the base query interface and RAM. *Note.* N@10, R@10, and M@10 denote NDCG@10, Recall@10, and MRR@10.

System	Method	N@10	R@10	M@10
AI-Scientist	Base	**86.2**	93.8	**83.6**
AI-Scientist	RAM	83.9	**95.4**	80.1
AgentLab	Base	86.2	93.8	83.6
AgentLab	RAM	**87.2**	**97.0**	**83.8**

Table 13: CodeRAG-Bench MBPP results comparing the base query interface and RAM. MBPP is already strong under the base query, so gains are smaller and less uniform than on ODEX. *Note.* N@10, R@10, and M@10 denote NDCG@10, Recall@10, and MRR@10.

E MLAgentBench Evaluation Details

Runtime environment. The runtime environment uses `LD_LIBRARY_PATH` pointing to the local MPI installation, `MACA_PATH`, `numpy==1.26.4`, and task-specific packages such as `tensorflow`, `dm-clrs`, `torch-geometric`, `evaluate`, `ogb`, and `absl-py`. Environment configuration is kept fixed across handoff methods within each system.

Evaluator layout. Several official MLAgentBench evaluators import helper code through relative paths. To preserve official evaluator semantics while scoring generated artifacts, we evaluate candidate outputs in a layout where official evaluator files are unchanged and relative imports resolve to the candidate implementation.

F PaperBench Six-Paper Subset

The main paper reports PaperBench code-only reproduction results on a controlled node < 200 six-paper subset. This subset is used to inspect paper-to-code handoff behavior under a fixed budget; it is not a full PaperBench leaderboard result.

Paper	Base	RAM	Δ
AP	23.4	26.8	+3.4
MU	62.2	74.3	+12.1
RC	30.9	32.3	+1.4
SNSE	39.2	51.2	+12.0
SoT-CFG	48.2	51.9	+3.7
SI	57.3	67.0	+9.7

Table 14: PaperBench code-only reproduction results on the node < 200 six-paper subset. *Note.* AP = adaptive-pruning; MU = mechanistic-understanding; RC = robust-clip; SNSE = sequential-neural-score-estimation; SoT-CFG = stay-on-topic-with-classifier-free-guidance; SI = stochastic-interpolants.

G Failure Analysis

Execution-grounded agent benchmarks evaluate complete artifact compliance, not only local code plausibility. Several runs create files or partial outputs but fail official evaluation because the generated artifact violates schema, shape, dependency, model-directory, or task-API contracts.

G.1 OGBN-Arxiv Dependency Rescue Check

We perform a dependency rescue check for OGBN-Arxiv after identifying that the official starter uses PyG `NeighborLoader`. We install `pyg_lib` and rerun the existing generated OGBN-Arxiv workspaces without regenerating code. This isolates the environment hypothesis from the handoff and generation outputs. The rescue check yields no additional official numeric scores. Logs indicate ABI incompatibility with the local Torch build and some generated outputs also violate the official prediction-shape contract. OGBN-Arxiv is therefore a useful failure-analysis case: a zero score can reflect environment compatibility and graph-output contracts, not only weak model reasoning.

Task	Failure type	Interpretation
Feedback	Artifact/schema contract	Some runs write `submission.csv`, but the evaluator expects six target columns: cohesion, syntax, vocabulary, phraseology, grammar, and conventions.
AMP-Parkinsons	Task API and submission shape	Generated fallbacks often produce empty or mismatched rating rows, causing SMAPE shape errors.
BabyLM	Output artifact contract	Evaluation requires a HuggingFace model directory under `output/` with `config.json`; generated scripts often fail to write a valid model artifact.
OGBN-Arxiv	Dependency and graph-prediction contract	`NeighborLoader` requires `pyg-lib` or `torch-sparse`; ABI mismatch and invalid `y_pred` shape prevent official scoring.
CLRS	Long-running training and API mismatch	Some runs score after timeout because checkpoints exist, while other runs fail on CLRS API or flag mismatches.

Table 15: Failure taxonomy for MLAgentBench strict official evaluation. These failures are retained instead of being converted into external-judge scores because they reflect artifact-contract requirements of execution-grounded benchmarks.