Read Less, Solve More: Token-Efficient Sparse Reading for AI Agents

Zedong Liu[1], **Jiaan Wu**[1], **Xinyang Ma**[1], **Le Xu**[1],
Kai Wang[2], **Yuanchao Hu**[2], **Dingwen Tao**[3], **Guangming Tan**[3]

[1]University of Chinese Academy of Sciences, Beijing, China
[2]Songshan Lake Materials Laboratory, Dongguan, China
[3]Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China

Abstract

Long-horizon agents increasingly rely on repeated access to external artifacts, yet current reading interfaces often expose entire objects even when only sparse evidence is needed. This over-reading increases token and latency costs and can dilute task-relevant evidence, while existing context-reduction methods mainly intervene after broad content has already entered the trajectory. We present *SparseRead*[1], a training-free, model-transparent reading layer that controls content admission before unnecessary evidence reaches the model context. SparseRead combines a regime-aware Read Gate, extensible Reader Backends, and a stateful protocol for bounded, source-anchored evidence acquisition with explicit refinement, verification, stopping, and fallback. Across six frontier models, including Claude Opus 5, and five workload scenarios, SparseRead reduces token volume by up to 92.9% and wall time by up to 89.0%, while preserving or improving task quality. Its consistent gains across three agent frameworks further demonstrate broad portability.

1 Introduction

Large language model (LLM) agents have extended the scope of language models from single-turn generation to long-horizon task execution (Yao et al. 2023; Liu et al. 2024a). By coupling reasoning with tool use and access to external artifacts, agents support increasingly complex applications across software engineering, scientific analysis, and interactive environments (Yang et al. 2024; Boiko et al. 2023; Zhou et al. 2024). This capability, however, introduces a markedly different inference profile: each step adds new observations that are repeatedly processed in subsequent model calls, causing token consumption, latency, and serving cost to grow rapidly with trajectory length.

Reading constitutes a major source of this growth. Prior work reports that external observations account for 60–80% of the tokens processed in agent trajectories (Wang et al. 2026). Our trace analysis reveals a sharper inefficiency within this budget: in representative high-sparsity tasks, only 6–15% of the exposed reading content contributes to the current decision. Paired executions further show that full-read agents can consume up to 17.3× more tokens without achieving higher task quality. Excess content is not necessarily benign; by diluting relevant evidence, it can induce erroneous tool use, source confusion, and failures to stop. We refer to this mismatch as *over-reading*: external artifacts are accessed at object granularity, although each decision often requires only a bounded evidence slice.

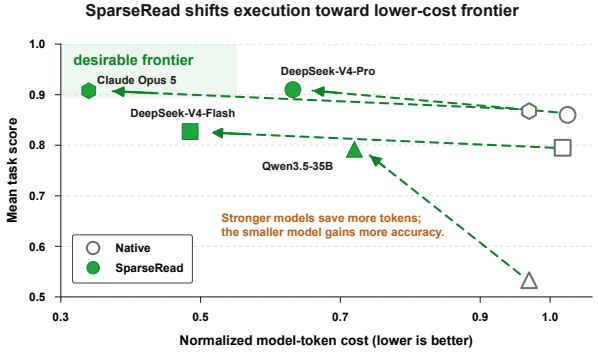

Figure 1: **Result teaser.** SparseRead shifts model execution toward a lower-cost, quality-preserving frontier.

Existing techniques reduce context cost without controlling the read itself. Long-context architectures and KV-cache compression reduce the compute and memory overhead of context, but cannot recover tokens already consumed or recover the reasoning trajectory drift (Zaheer et al. 2020; Li et al. 2024). Model-based compression introduces additional inference overhead, while retrieval is often detached from the agent's evolving information needs (Guu et al. 2020). ACON and Complexity Trap reduce tool observations but without explicit control over artifact reading (Kang et al. 2026; Lindenbauer et al. 2025). A general mechanism is still needed to govern artifact access before unnecessary content enters the agent context.

Human reading provides a natural abstraction for this missing capability. Readers first inspect the structure of a long artifact, concentrate on the passages relevant to their current purpose, revisit only unresolved details, and stop once the information need has been satisfied. We call this capability *Sparse Reading*: the agent reads only what its current reasoning requires and expands the read only when additional evidence is needed.

Turning this pattern into an agent mechanism raises *two challenges*. (i) The Reader must provide clear evidence for the

[1]https://github.com/Zedong-Liu/SparseReading

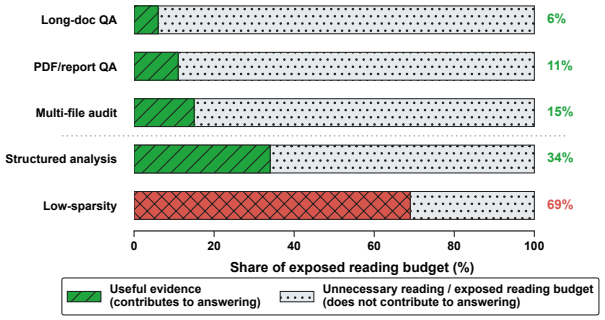

Figure 2: Reading-budget breakdown. In high-sparsity scenarios, useful evidence occupies only a small fraction of the exposed broad content.

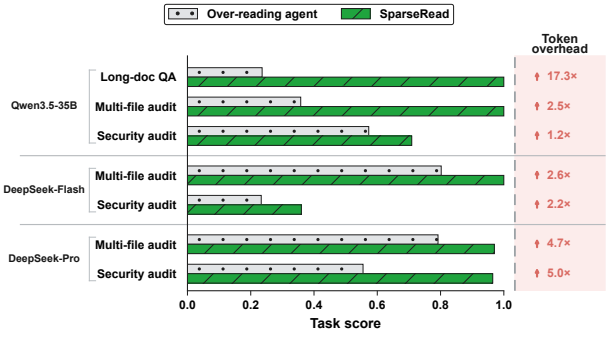

Figure 3: Quality-efficiency comparison. Sparse reading matches or improves task quality while using fewer tokens than full reading. More exposed context does not automatically produce better answers.

agent to assess whether further reading is needed. (ii) Sparse Reading also requires a regime boundary because its benefits are not universal. When sparsity is low, the achievable savings cannot justify the overhead. This calls for explicit evidence state and online regime control.

We present SparseRead, a framework that controls artifact admission before broad content enters the model context. Its architecture combines a stateful Sparse Reading protocol, extensible Reader Backends, and a Read Gate that selects the operating mode online during inference. SparseRead can be integrated into existing agent frameworks and used with existing LLMs. It requires no changes to model parameters, no additional training, and no auxiliary compression model.

This paper makes the following contributions:

- **Stateful Sparse Reading protocol.** We introduce a cooperative protocol that turns broad artifact access into a bounded and stateful reading process. By making the progress and completion of each partial read explicit, it establishes the trust needed to avoid repeated broad reading.
- **Extensible Reader Backends.** We separate reading control from evidence acquisition through a unified Reader interface. This design extends the same protocol across heterogeneous artifacts and specialized domains while keeping the agent-facing interface unchanged.
- **Regime-aware Read Gate.** We introduce an inference-time multi-mode controller that dynamically determines how strongly SparseRead should intervene. It activates Sparse Reading only when beneficial and preserves native reading beyond its effective regime.
- **Comprehensive evaluation.** Across six models and five scenarios, SparseRead reduces token use by up to 92.9% and job time by up to 89.0%, while preserving or improving task quality. It achieves the lowest token and time costs in every comparison with SOTA baselines and transfers across three agent frameworks.

2 Quantifying the Cost of Over-Reading

We first quantify how much admitted reading is actually useful, then show that exposing more content does not reliably improve task quality. Together, these observations make over-reading a measurable failure mode, not only a cost problem.

2.1 Useful Evidence Is Sparse

Figure 2 breaks down trajectory-level reading budgets by scenario. In high-sparsity settings such as long-document QA, PDF QA, and audit bundles, the useful evidence occupies only a small share of the reading budget (6-34%), rendering full reading highly wasteful.

Conventional post-read compression cannot solve this problem. A summarizer shortens content only after it has already been fully admitted into context, meaning the expensive broad read has already been paid. More critically, it fails to prevent the irreversible trajectory drift induced by over-reading, an effect that severely degrades performance in long-horizon tasks. Sparse Reading instead intercepts before ingestion, extracting only task-relevant evidence and avoiding the full-read overhead altogether.

2.2 More Context Does Not Improve Task Quality

If broad reading always improved quality, over-reading would be a cost-only problem. Figure 3 shows otherwise. In paired high-sparsity runs, full-read agents spend up to 17.3 times more tokens than SparseRead while failing to improve task score. Across model and task groups, selective reading preserves or improves score while reducing exposed context. The result does not imply that less context is always better; it shows that broad context is not a free substitute for reading control.

Current agents have tools for reading and compression, but they lack a first-class mechanism for deciding how much of an artifact should be admitted before the next decision. We use *Sparse Reading* to name this missing capability. Sparse Reading is a pre-read evidence acquisition strategy for agents: before a large external artifact is fully exposed to model context, the system identifies and exposes task-relevant evidence for the current step, preserves source anchors and unresolved needs, and maintains a fallback path to native reading.

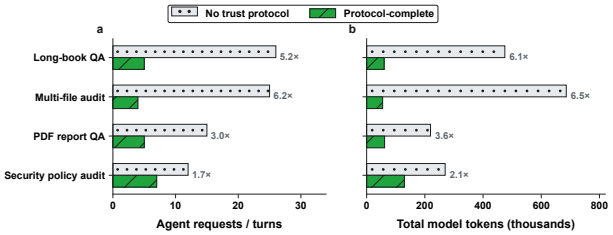

Figure 4: Protocol sufficiency challenge. Compression alone is not enough. Without state that lets the model trust, an agent can repeatedly negotiate the same read and consume many more requests and tokens.

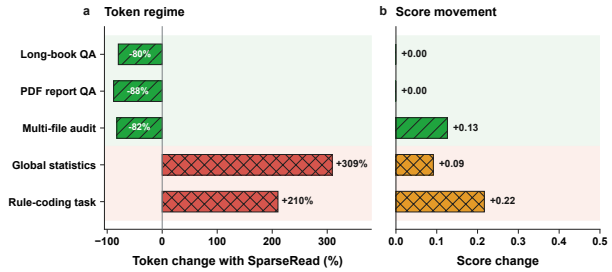

Figure 5: Regime boundary. Sparse Reading is most useful when the task has a sparse useful-content structure. Low-sparsity or native-fit tasks can pay protocol overhead, motivating selective intervention rather than an always-on reader.

3 Why Compression Alone Is Not Enough

Problem formulation. At step t, let o denote the current external artifact. The core challenge is to control the content x_t read by the model before the next decision: filtering out irrelevant exposure while preserving critical information integrity and task quality, ensuring no significant degradation in the model's ultimate performance.

Compression Alone. A naive approach to this problem is to apply a reading compressor. Before fully ingesting a large artifact, the system processes it through a compressor to yield a more compact observation. While this significantly reduces the ingested content, it fails to ground a reliable reading process. Our experiments reveal two failure modes: (1) missing provenance and compression rationale fails to engender the model trust required to terminate broad reading, and (2) indiscriminate compression disrupts tasks that demand extensive computation or exact native access.

3.1 Challenge 1: Trust Between Model and Reading Compressor

Figure 4 shows the cost of lacking a protocol. Without knowing the coverage of the sparse result, agents require 1.7–6.2× more requests and 2.1–12.5× more tokens. The failure is not that compressed evidence is always too small. The failure is that the agent cannot tell whether the evidence is complete or unsafe to use.

We refer to this failure as *compression distrust*. The agent receives a short answer but lacks source anchors, unresolved requirements, and a bounded next action. It may repeatedly re-read the source or expand its search. If the system blindly intercepts and compresses again, it triggers an endless loop of sparse extraction, distrust, broad reread, and another underspecified result.

Sparse Reading therefore needs a trust protocol, not only a compressor. Section 4 shows how SparseRead addresses this by maintaining a compact yet explicit protocol state.

3.2 Challenge 2: The Regime Boundary of Sparse Reading

Sparse Reading is well-suited for tasks that require extracting localized information from large artifacts. Long-document QA, PDF comprehension, and audit bundles exhibit this characteristic. However, some tasks favor the native path: they require whole-table computation, exact scanning, or direct small-file inspection. For these tasks, sparse reading may add overhead without reducing token usage.

Figure 5 illustrates this boundary. In long-document QA, Sparse Reading reduces token consumption by 80–88%, while simultaneously improving quality in multi-file audit tasks. Although Low-sparsity controls can still improve score, the token cost is unacceptable: forced sparse reading increases tokens by 309.2% on T67 and 210.5% on T59. Therefore, sparse reading should not be universally enforced; instead, a safe application boundary is required.

Design implication. Together, these two challenges show that Sparse Reading needs more than a compressor. It needs compact protocol state to eliminate compression distrust and prevent loops of blind re-reading. It also needs an online decision about whether the sparse path should be used for the current artifact and step.

4 SparseRead Design

SparseRead organizes large-object reading into two independently extensible planes. The control plane exposes a reading protocol to the agent, maintains model-visible read state, and regulates context admission. The evidence-compression plane performs evidence acquisition through Reader Backends. This separation allows a new Reader to register its capabilities without expanding the agent-facing macro-action set or modifying the admission policy. The model interface therefore remains compact as backend coverage grows. Figure 6 shows the resulting architecture.

The Unified Reader Interface fixes the request, action, and evidence contract between the two planes. The control plane depends only on these stable semantics, while each Reader encapsulates object- or domain-specific access. Reading policy and evidence-acquisition capability can therefore evolve without propagating backend changes into the agent interface.

SparseRead compresses a read result before admitting it into context and leaves the conversation and tool prefix preceding the read boundary unchanged. Under exact-match prefix caching, cache entries ending at or before that boundary remain reusable, while the compressed observation shortens the appended suffix. SparseRead thus reduces the prefill cost

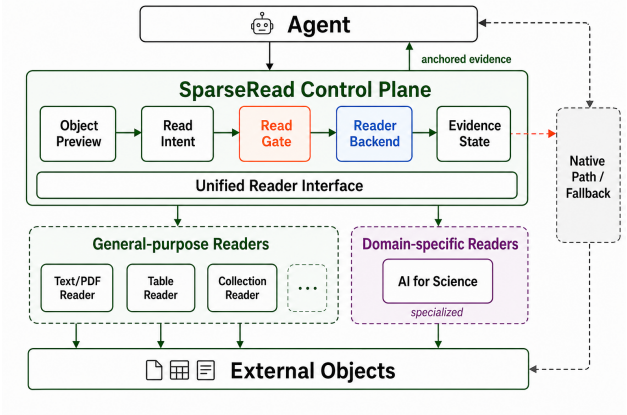

Figure 6: SparseRead separates the agent-facing control plane from replaceable Reader Backends. The Read Gate selects the intervention mode, while the protocol governs evidence acquisition, state updates, and fallback.

of new content while preserving the prefix reuse available before the read.

4.1 Agent-Facing Reading Protocol

SparseRead formulates large-object reading as sequential evidence acquisition under partial observability. At the entry of a read request, an Object Preview gives the agent a bounded view from which it decides what evidence is needed. Subsequent results accumulate in a model-visible Evidence State. The agent is the explicit policy actor; the control plane validates actions, enforces budgets, and updates state.

The agent writes a Read Intent from its task state and the Object Preview, and may narrow it in later rounds. The intent is a compact structured field in the ordinary read action. It introduces no separate auxiliary model, so its control overhead is small relative to reading the object, yet it determines the evidence sought by the Reader. The Evidence State organizes returned results as a verifiable read record and preserves the current search boundary. The agent can therefore distinguish evidence absent from an inspected region from evidence in a region not yet examined, and decide whether to continue, verify, or stop.

The protocol separates semantic reading decisions from mechanical tool selection through a stable macro-action set $\mathcal{A} = \{\textsc{scout}, \textsc{focus}, \textsc{collect}, \textsc{refine}, \textsc{verify}\}$. The agent explicitly selects a macro action. The Reader Router then chooses a compatible backend using the object, intent, and current read state. The selected Reader expands the macro action into object-specific operations and returns anchored observations. Adding a Reader therefore does not add another tool choice for the agent. Algorithm 1 summarizes this.

Closure is defined relative to the Read Intent. A read stops when accumulated evidence supports the pending decision and every residual need has been resolved, judged non-critical, or explicitly accepted as a risk by the agent. The agent may narrow the intent in later rounds, but $\mathrm{AgentRefineIntent}$ preserves unresolved obligations from the original request. An empty routing result, stalled progress, or budget exhaustion triggers protocol-level native fallback.

Algorithm 1: Agent-Facing Sparse Reading Protocol

Input: artifact o, agent state z_0, native access N
Parameters: round budget K, macro-action set $\mathcal{A}$
$p \leftarrow \mathrm{Preview}(o)$, $E \leftarrow \varnothing$, $z \leftarrow z_0$
$h \leftarrow \mathrm{AgentIntent}(z, p, E)$
$d \leftarrow \mathrm{ReadGate}(p, h)$
if $d = \textsc{native}$ **then**
  return $N(o)$
end if
if $d = \textsc{advisory}$ **and** $\mathrm{AgentNative}(z, p, h)$ **then**
  return $N(o)$
end if
for $t = 1, \ldots, K$ **do**
  $a \leftarrow \mathrm{AgentSelect}(z, h, E; \mathcal{A})$
  $R \leftarrow \mathrm{ReaderRouter}(p, h, E, a)$
  if $R = \varnothing$ **then**
    return $\mathrm{Fallback}(N, o, E)$
  end if
  $x \leftarrow \mathrm{Read}_R(o, a, h, E)$
  $E' \leftarrow \mathrm{Update}(E, x)$
  if $\mathrm{AgentClosed}(E', h)$ **then**
    return E'
  end if
  if $\mathrm{Unusable}(x)$ **or** $\mathrm{Stalled}(E, E')$ **then**
    return $\mathrm{Fallback}(N, o, E')$
  end if
  $z \leftarrow \mathrm{AgentUpdate}(z, E')$
  $E \leftarrow E'$
  $h \leftarrow \mathrm{AgentRefineIntent}(h, z, p, E)$
end for
return $\mathrm{Fallback}(N, o, E)$

4.2 Extensible Reader Backends

Each Reader registers the object classes and macro actions it supports, together with its profiled execution cost. Given (p, h, E, a), the Reader Router first filters out backends that cannot handle the current object or action. It ranks the remaining Readers by how specifically their capabilities match the Read Intent and uses estimated execution cost to break ties. A domain-specific Reader can therefore override a general-purpose Reader through a more specific capability match while preserving the same protocol entry point.

The five macro actions retain stable semantics across Readers: SCOUT maps the information space, FOCUS locates bounded evidence, COLLECT satisfies multiple evidence needs, REFINE resolves remaining gaps, and VERIFY checks an existing claim. A Reader may implement only a subset of these actions and declares that subset at registration. Each implementation expands a macro action into object-specific operations and normalizes the result into an anchored observation before the control plane updates the Evidence State.

The Readers evaluated in this work invoke no additional learned model during backend execution, keeping read-time overhead low and predictable. The interface also admits model-based pruning methods. SWE-Pruner (Wang et al. 2026) and Squeez (Kovács 2026), for example, can be wrapped as typed Readers by mapping their task-conditioned outputs to the same evidence contract. Such integration is an

Algorithm 2: Regime-Aware Read Gate

Input: preview p, intent h
Parameters: weights w_s, w_c, w_r, thresholds $\theta_n < \theta_f$
if $\neg\text{Supported}(p, h)$ **or** $\text{RequiresNative}(p, h)$ **then**
    return NATIVE
end if
$(s, c, r) \leftarrow \text{NormalizedSignals}(p, h)$
$b \leftarrow w_s s - w_c c - w_r r$
if $b \geq \theta_f$ **then**
    return FORCE
end if
if $b \leq \theta_n$ **then**
    return NATIVE
end if
return ADVISORY

extensibility path and is not evaluated in this work.

The same registration mechanism supports both general-purpose and domain-specific Readers. General-purpose Readers cover common object classes, whereas specialized Readers may encode domain ontologies, operators, and evidence-validity rules. An AI-for-Science Reader, for example, can use scientific entities and validation constraints to return traceable evidence with domain semantics.

4.3 Regime-Aware Read Gate

Following the regime boundary in Section 3.2, the Read Gate performs one request-level routing decision. Offline profiling showed that observable pre-read signals, including object scale, intent scope, and Reader cost, were sufficient to separate requests with clear sparse-path benefit from those favoring native access. We therefore use a deterministic Gate and assign boundary cases to ADVISORY, avoiding the inference and calibration overhead of a learned router.

The Gate first applies hard constraints. A request without a compatible Reader, or whose semantics require complete native execution, selects NATIVE. For remaining requests, the Gate obtains normalized estimates of avoidable context exposure s, protocol cost c, and omission risk r, and computes

$$b(p, h) = w_s s - w_c c - w_r r, \qquad w_s, w_c, w_r > 0.$$

The weights and thresholds are selected jointly on a development workload to maximize token reduction subject to a predefined quality-loss tolerance. At runtime, the Gate evaluates the linear margin once: $b \geq \theta_f$ selects FORCE, $b \leq \theta_n$ selects NATIVE, and $\theta_n < b < \theta_f$ selects ADVISORY. Algorithm 2 gives the complete rule.

The three outputs differ only in intervention strength. FORCE requires the request to enter the sparse protocol before broad native access while retaining protocol-level fallback. ADVISORY keeps both paths available and lets the agent choose. NATIVE bypasses the sparse protocol. The Gate terminates after this entry decision. The agent selects all subsequent macro actions, while Reader routing, closure, and fallback remain control-plane responsibilities.

5 Implementation and Agent Integration

Agent harness integration. The SparseRead core is a shared runtime: it owns the preview/read protocol, typed readers, gate policy, artifact state, and bridge server. Each agent harness adds only the framework-specific surface needed to route broad object access into that runtime. NanoBot registers the SparseRead tools directly. OpenCode and OpenClaw use thin plugin or bridge adapters that expose `sro_preview` and `sro_read`, retain `sro_card` for compatibility, and translate local read/open events into the same core calls. Model-facing guidance such as `SKILL.md` tells the agent when to start from preview, when to issue a targeted read, and when to return to native access. This keeps the control plane in the core and leaves framework code as a small adapter layer.

Reader backends and state. The implementation includes typed readers for text/PDF, structured tables, multi-file collections, and script-library style artifacts. Text/PDF readers extract page or heading anchors and bounded local windows. Table readers expose schema, samples, row/column projections, and calculation-ready TSV fragments. Collection readers build an inventory before selecting candidate files. Each backend receives an Object Preview, Read Intent, and previous Evidence State, then returns anchored evidence with coverage, unresolved requirements, status, slot digests.

6 Evaluation

6.1 Experimental Setup

The evaluation addresses four research questions. **RQ1** examines whether SparseRead improves reading efficiency across models and scenarios without sacrificing task quality. **RQ2** compares SparseRead with existing context-reduction methods in both token cost and end-to-end latency. **RQ3** studies whether the Read Gate intervenes selectively across different task regimes. **RQ4** evaluates whether the same SparseRead design transfers across agent frameworks.

Tasks and models. The general evaluation draws 125 tasks from QwenClawBench, Claw-Eval, PinchBench, and LooGLE (Li et al. 2023a; Ye et al. 2026), covering long-context reading, multi-file audit and diagnosis, structured analysis, and native-fit controls. We further evaluate materials-chemistry tasks as an AI for Science scenario. The main NanoBot experiment includes six models: Claude Opus 5, Qwen3.6-Plus, DeepSeek-V4-Flash, DeepSeek-V4-Pro, GLM-5.1, and Kimi-K2.5. Opus 5 is the strongest model evaluated and allows us to test whether sparse reading remains useful as model capability increases.

Agent frameworks. We integrate SparseRead into *NanoBot v0.2.0, OpenCode v1.17.14,* and *OpenClaw v2026.6.11*. All integrations share the same reading protocol and Reader Backends, while thin adapters connect the protocol to each framework's tool interface. The cross-framework experiment uses five models and requires no model retraining.

Baselines and metrics. We compare against (i) Naive full reading, (ii) the model-based *ACON* zero-shot compressor , and (iii) observation masking from *Complexity Trap* (Obs. masking), configured to retain the latest ten observations (Kang et al. 2026; Lindenbauer et al. 2025). All costs are measured end to end, including the auxiliary model calls made by ACON. We report total token reduction, task-score change, wall-time saving, and, for the gate analysis, request-count change averaged over three runs. Reductions and score

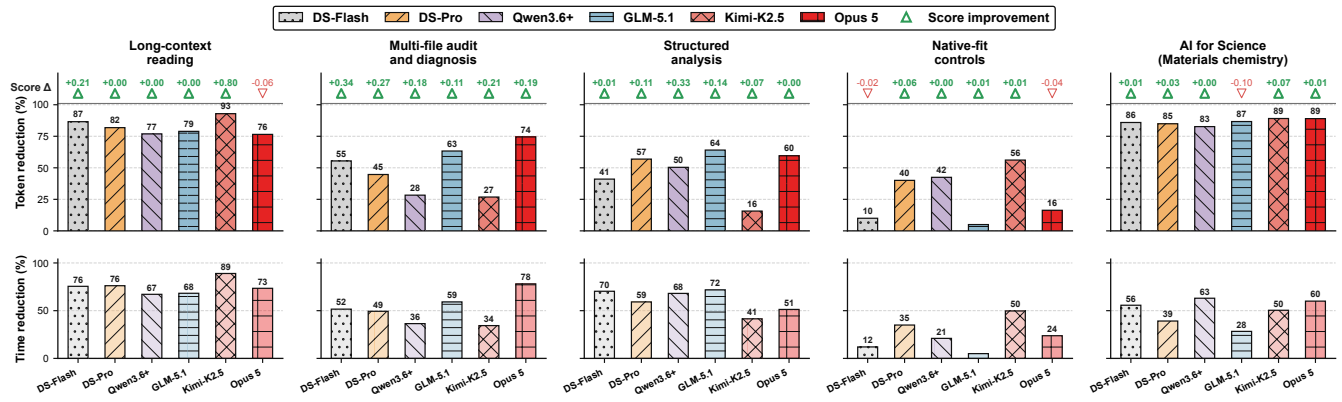

Figure 7: SparseRead results across six models and five scenarios on NanoBot. The upper panels report total-token reduction relative to Naive, with markers showing the corresponding score delta. The lower panels report end-to-end wall-time reduction. Positive bars indicate lower cost.

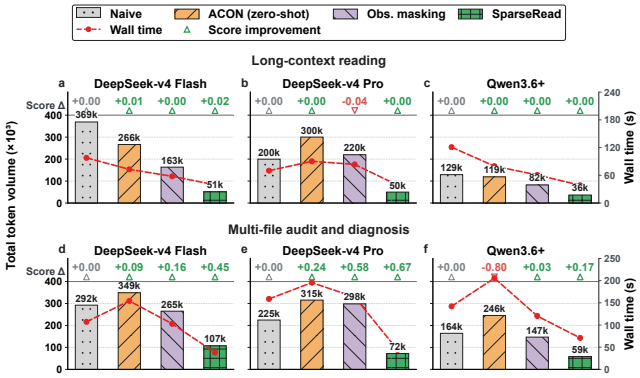

Figure 8: Comparison with context-reduction baselines across two scenarios and three models.

deltas are computed relative to Naive; a positive delta indicates an improvement.

6.2 Effectiveness Across Models and Scenarios

Figure 7 shows consistent gains across the full evaluation matrix. SparseRead reduces both token volume and wall time in all 30 model–scenario cells, with reductions of up to 92.9% and 89.0%, respectively; 26 cells preserve or improve task score. Across the four sparse-fit scenarios, token reduction ranges from 15.7% to 92.9% and wall-time saving from 28.3% to 89.0%, with non-negative score changes in 22 of 24 cells. The four isolated changes between -0.10 and -0.02 exhibit no consistent model-level or scenario-level pattern and are consistent with ordinary stochastic variation in agent execution.

The gain persists for Opus 5, the strongest model evaluated, which saves 59.6–89.0% of tokens and 51.3–78.1% of wall time across the sparse-fit scenarios. Thus, stronger model capability does not remove the reading bottleneck. The same protocol also transfers across long documents, multi-file repositories, structured artifacts, native-fit controls, and scientific workloads. In AI for Science, all six models reduce token use by 82.7–89.1% and wall time by 28.3–63.0%, showing that SparseRead is neither model-specific nor scenario-specific.

Table 1: End-to-end SparseRead results across three agent frameworks and five models.

Framework	Model	Token red.	Score Δ	Time save
NanoBot	Qwen3.6-Plus	69.0%	+0.039	64.4%
	DeepSeek-Flash	22.4%	+0.140	25.6%
	DeepSeek-Pro	71.3%	−0.015	73.1%
	GLM-5.1	61.0%	+0.012	52.6%
	Kimi-K2.5	92.9%	+0.598	85.4%
OpenCode	Qwen3.6-Plus	25.2%	+0.333	22.8%
	DeepSeek-Flash	71.8%	+0.143	64.9%
	DeepSeek-Pro	58.5%	−0.013	62.9%
	GLM-5.1	75.4%	+0.033	68.9%
	Kimi-K2.5	83.7%	−0.031	73.7%
OpenClaw	Qwen3.6-Plus	46.7%	+0.032	56.8%
	DeepSeek-Flash	28.7%	+0.062	26.5%
	DeepSeek-Pro	14.5%	+0.151	28.2%
	GLM-5.1	28.7%	+0.118	38.6%
	Kimi-K2.5	42.8%	+0.032	24.6%

6.3 Comparison with Context-Reduction Baselines

Figure 8 compares SparseRead with ACON and CT observation masking across six model–scenario settings. SparseRead achieves the lowest token volume and wall time in every setting and is the only method whose score never falls below Naive. It reduces tokens by 63.4–86.3% and wall time by 44.7–72.3%, reaching maximum savings of 86.3% and 72.3%. The measured wall-time reductions correspond to a 1.8–3.6× end-to-end completion-time speedup, confirming that compression yields substantial execution acceleration.

ACON exceeds Naive in both token use and wall time in four settings and loses 0.80 score on Qwen3.6-Plus multi-

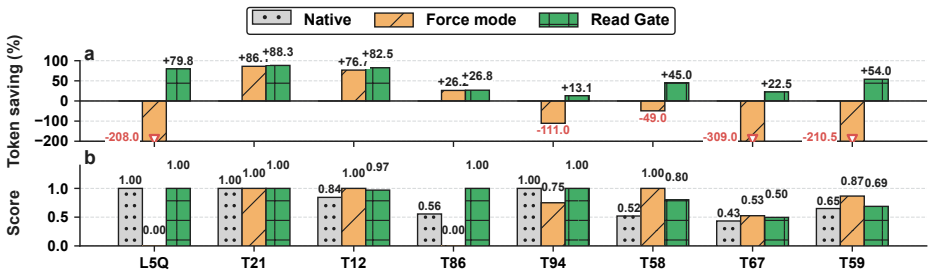

Figure 9: Read-Gate behavior across 8 matched task on Deepseek-v4 Pro. Gray bars denote Native, and colored bars denote SparseRead under the gate-selected policy. Annotations report score changes or cost reductions relative to Naive.

file diagnosis. This instability is consistent with the overhead of auxiliary model calls and the absence of an explicit intent–action–evidence contract. SparseRead instead provides a structured narrowing and stopping procedure. CT's fixed last-ten observation mask is position-based and insensitive to evidence demand; in both DeepSeek-V4-Pro settings, it consumes more tokens and time than Naive. These results illustrate the advantage of protocol-guided evidence selection over generic generation-based compression or absolute context masking.

6.4 Portability Across Agent Frameworks

Table 1 shows that SparseRead reduces token volume and wall time in all 15 framework–model cells across NanoBot, OpenCode, and OpenClaw, while preserving or improving score in 12. Median token reductions are 69.0%, 71.8%, and 28.7%, and median wall-time savings are 64.4%, 64.9%, and 28.2%, respectively. The three small score changes, ranging from -0.031 to -0.013, are consistent with trajectory-level variation. Reusing the same protocol and Reader Backends through thin adapters demonstrates practical plug-in portability across substantially different agent frameworks without model retraining or scenario-specific redesign.

6.5 Selective Intervention by the Read Gate

Figure 9 evaluates the Read Gate across 8 matched tasks on DeepSeek-Pro. SparseRead reduces token cost in 38 pairs and preserves or improves score in 39; all 14 clear-win cases satisfy both conditions, with a median token reduction of 79.6%. Among the 24 gate/pass cases, 20 reduce token cost and 23 preserve or improve score, while request counts generally follow the token trend. The few exceptions concentrate in low-sparsity tasks and show no systematic quality degradation, indicating that the gate captures strong sparse-reading opportunities while remaining conservative near the sparse/-native boundary.

7 Related Work

Context / KV-cache compression. Selective Context, the LLMLingua family, Gist Tokens, AutoCompressors, and ICAE compress an assembled context, often through additional inference or training (Li et al. 2023b; Jiang et al. 2023, 2024; Pan et al. 2024; Mu, Li, and Goodman 2023; Chevalier et al. 2023; Liu et al. 2026b,a), whereas H_2O, Scissorhands, StreamingLLM, SnapKV, PyramidKV, KIVI, and CacheGen optimize KV states during inference (Zhang et al. 2023; Xiao et al. 2024; Li et al. 2024; Cai et al. 2024; Liu et al. 2024b; He et al. 2024). Because they intervene after content has been read, they cannot recover the cost or long-horizon trajectory drift already induced by over-reading. SparseRead selects evidence before admission without rewriting the existing prefix, preserving prefix-cache reuse (Gim et al. 2024) while remaining complementary to KV optimization.

Agent observation compression. SWE-Pruner, Squeez, CoACT, AgentDiet, and TACO compress agent observations or trajectories, but primarily target code and terminal settings, with several requiring specialized training or online adaptation (Wang et al. 2026; Kovács 2026; Chen et al. 2026; Xiao et al. 2026; Ren et al. 2026). ACON uses generative models to compress general observations and histories, incurring auxiliary inference without an explicit stateful reading protocol (Kang et al. 2026); training-free observation masking instead follows a fixed positional rule (Lindenbauer et al. 2025). Unlike iterative retrieval methods that search indexed corpora, SparseRead incrementally reads already-accessible heterogeneous objects through stateful control (Asai et al. 2024). SparseRead coordinates extensible Readers through one protocol without an auxiliary compressor; specialized methods can be incorporated as Reader Backends.

8 Conclusion

SparseRead moves agent efficiency from post-hoc context compression to *pre-reading control* over external objects. Its stateful protocol, extensible Readers, and dynamic Read Gate provide a model-independent and prefix-cache-friendly abstraction that transfers across heterogeneous artifacts and agent frameworks. The gains persist as model capability increases, suggesting that over-reading is not merely a weakness of current models, but a structural inefficiency in how agents interact with external information. Beyond improving existing agents, SparseRead points to a broader design principle for production-grade agent harnesses: reading should be an explicit, controllable part of the execution path rather than an unrestricted tool side effect. This perspective may also inform future model and agent training, where learning *what to read, how much to read, and when to stop* can become a first-class capability alongside reasoning and tool use.

References

Asai, A.; Wu, Z.; Wang, Y.; Sil, A.; and Hajishirzi, H. 2024. Self-rag: Learning to retrieve, generate, and critique through self-reflection. In *International conference on learning representations*, volume 2024, 9112–9141.

Boiko, D. A.; MacKnight, R.; Kline, B.; and Gomes, G. 2023. Autonomous Chemical Research with Large Language Models. *Nature*, 624(7992): 570–578.

Cai, Z.; Zhang, Y.; Gao, B.; Liu, Y.; Li, Y.; Liu, T.; Lu, K.; Xiong, W.; Dong, Y.; Hu, J.; and Xiao, W. 2024. PyramidKV: Dynamic KV Cache Compression Based on Pyramidal Information Funneling. *arXiv preprint arXiv:2406.02069*.

Chen, H.; Zhu, Y.; Zhang, Y.; and Li, J. 2026. CoACT: Action-Preserving Observation Compression for Coding Agents. *arXiv preprint arXiv:2607.02911*.

Chevalier, A.; Wettig, A.; Ajith, A.; and Chen, D. 2023. Adapting Language Models to Compress Contexts. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 3829–3846. Association for Computational Linguistics.

Gim, I.; Chen, G.; Lee, S.-s.; Sarda, N.; Khandelwal, A.; and Zhong, L. 2024. Prompt Cache: Modular Attention Reuse for Low-Latency Inference. In *Proceedings of Machine Learning and Systems*, volume 6, 325–338.

Guu, K.; Lee, K.; Tung, Z.; Pasupat, P.; and Chang, M. 2020. Retrieval Augmented Language Model Pre-Training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, 3929–3938. PMLR.

He, Y.; Zhang, L.; Wu, W.; Liu, J.; Zhou, H.; and Zhuang, B. 2024. ZipCache: Accurate and Efficient KV Cache Quantization with Salient Token Identification. In *Advances in Neural Information Processing Systems*, volume 37, 68287–68307.

Jiang, H.; Wu, Q.; Lin, C.-Y.; Yang, Y.; and Qiu, L. 2023. LLMLingua: Compressing Prompts for Accelerated Inference of Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 13358–13376. Association for Computational Linguistics.

Jiang, H.; Wu, Q.; Luo, X.; Li, D.; Lin, C.-Y.; Yang, Y.; and Qiu, L. 2024. LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 1658–1677. Association for Computational Linguistics.

Kang, M.; Chen, W.-N.; Han, D.; Inan, H. A.; Wutschitz, L.; Chen, Y.; Sim, R.; and Rajmohan, S. 2026. ACON: Optimizing Context Compression for Long-horizon LLM Agents. In *International Conference on Machine Learning*.

Kovács, Á. 2026. Squeez: Task-Conditioned Tool-Output Pruning for Coding Agents. *arXiv preprint arXiv:2604.04979*.

Li, J.; Wang, M.; Zheng, Z.; and Zhang, M. 2023a. Loogle: Can long-context language models understand long contexts? *arXiv preprint arXiv:2311.04939*.

Li, Y.; Dong, B.; Guerin, F.; and Lin, C. 2023b. Compressing Context to Enhance Inference Efficiency of Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 6342–6353. Association for Computational Linguistics.

Li, Y.; Huang, Y.; Yang, B.; Venkitesh, B.; Locatelli, A.; Ye, H.; Cai, T.; Lewis, P.; and Chen, D. 2024. SnapKV: LLM Knows What You are Looking for Before Generation. In *Advances in Neural Information Processing Systems*, volume 37, 22947–22970.

Lindenbauer, T.; Slinko, I.; Felder, L.; Bogomolov, E.; and Zharov, Y. 2025. The Complexity Trap: Simple Observation Masking Is as Efficient as LLM Summarization for Agent Context Management. *arXiv preprint arXiv:2508.21433*.

Liu, X.; Yu, H.; Zhang, H.; Xu, Y.; Lei, X.; Lai, H.; Gu, Y.; Ding, H.; Men, K.; Yang, K.; et al. 2024a. Agentbench: Evaluating llms as agents. In *International Conference on Learning Representations*, volume 2024, 52989–53046.

Liu, Z.; Cheng, S.; Tan, G.; You, Y.; and Tao, D. 2026a. Elasticmm: Efficient multimodal llms serving with elastic multimodal parallelism. *Advances in Neural Information Processing Systems*, 38: 94264–94289.

Liu, Z.; Ma, X.; Luo, D.; Zhao, H.; Lu, B.; Huang, W.; Gu, Y.; Liu, X.; Wei, Z.; Liu, J.; et al. 2026b. KVServe: Service-aware kv cache compression for communication-efficient disaggregated LLM serving. In *Proceedings of the ACM SIGCOMM 2026 Conference*, 1–15.

Liu, Z.; Yuan, J.; Jin, H.; Zhong, S.; Xu, Z.; Braverman, V.; Chen, B.; and Hu, X. 2024b. KIVI: A Tuning-Free Asymmetric 2bit Quantization for KV Cache. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, 32332–32344. PMLR.

Mu, J.; Li, X. L.; and Goodman, N. 2023. Learning to Compress Prompts with Gist Tokens. In *Advances in Neural Information Processing Systems*, volume 36, 19327–19352.

Pan, Z.; Wu, Q.; Jiang, H.; Xia, M.; Luo, X.; Zhang, J.; Lin, Q.; Rühle, V.; Yang, Y.; Lin, C.-Y.; Zhao, H. V.; Qiu, L.; and Zhang, D. 2024. LLMLingua-2: Data Distillation for Efficient and Faithful Task-Agnostic Prompt Compression. In *Findings of the Association for Computational Linguistics: ACL 2024*, 963–981. Association for Computational Linguistics.

Ren, J.; Wu, S.; Li, Y.; Zhu, K.; Xu, S.; Feng, B.; Yuan, R.; Zhang, W.; Batista-Navarro, R.; Yang, J.; and Lin, C. 2026. A Self-Evolving Framework for Efficient Terminal Agents via Observational Context Compression. *arXiv preprint arXiv:2604.19572*.

Wang, Y.; Shi, Y.; Yang, M.; Zhang, R.; He, S.; Lian, H.; Chen, Y.; Ye, S.; Cai, K.; and Gu, X. 2026. SWE-Pruner: Self-Adaptive Context Pruning for Coding Agents. *arXiv preprint arXiv:2601.16746*.

Xiao, G.; Tian, Y.; Chen, B.; Han, S.; and Lewis, M. 2024. Efficient Streaming Language Models with Attention Sinks. In *International Conference on Learning Representations*.

Xiao, Y.-A.; Gao, P.; Peng, C.; and Xiong, Y. 2026. Reducing Cost of LLM Agents with Trajectory Reduction. *Proceedings of the ACM on Software Engineering*, 3(FSE): 1241–1263.

Yang, J.; Jimenez, C. E.; Wettig, A.; Lieret, K.; Yao, S.; Narasimhan, K.; and Press, O. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *Advances in Neural Information Processing Systems*, volume 37, 50528–50652.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K. R.; and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations*. OpenReview.net.

Ye, B.; Li, R.; Yang, Q.; Liu, Y.; Yao, L.; Lv, H.; Xie, Z.; An, C.; Li, L.; Kong, L.; et al. 2026. Claw-Eval: Towards Trustworthy Evaluation of Autonomous Agents. *arXiv preprint arXiv:2604.06132*.

Zaheer, M.; Guruganesh, G.; Dubey, K. A.; Ainslie, J.; Alberti, C.; Ontanon, S.; Pham, P.; Ravula, A.; Wang, Q.; Yang, L.; and Ahmed, A. 2020. Big Bird: Transformers for Longer Sequences. In *Advances in Neural Information Processing Systems*, volume 33, 17283–17297. Curran Associates, Inc.

Zhang, Z.; Sheng, Y.; Zhou, T.; Chen, T.; Zheng, L.; Cai, R.; Song, Z.; Tian, Y.; Ré, C.; Barrett, C.; Wang, Z.; and Chen, B. 2023. H2O: Heavy-Hitter Oracle for Efficient Generative Inference of Large Language Models. In *Advances in Neural Information Processing Systems*, volume 36, 34661–34710.

Zhou, S.; Xu, F. F.; Zhu, H.; Zhou, X.; Lo, R.; Sridhar, A.; Cheng, X.; Ou, T.; Bisk, Y.; Fried, D.; et al. 2024. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, 15585–15606.