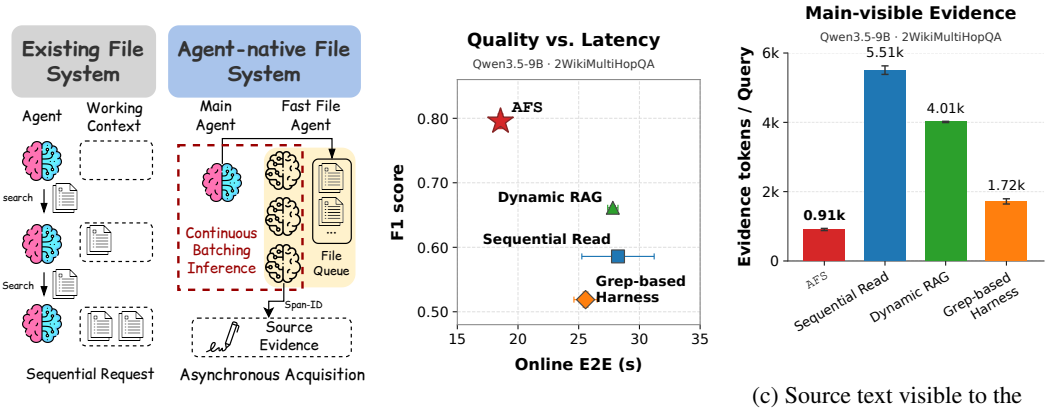
AFS: TOWARDS AN AGENT-NATIVE FILE SYSTEM

Anonymous authors
Paper under double-blind review

ABSTRACT

AI agents increasingly operate in file-rich workspaces that extend beyond a single context window. However, conventional file system interfaces are designed for humans, whose way of perceiving and navigating files differs fundamentally from that of LLMs. This mismatch severely limits the efficiency and speed of agent file access. We introduce AFS, an agent-native file system with three complementary designs. First, it uses offline preprocessing to organize files into identifiable source spans and builds reusable per-file key–value (KV) caches. Second, several parallel *File Agents* use continuous batching to inspect multiple files efficiently through shared model forward passes. Third, AFS decouples file inspection from reasoning, allowing *File Agents* to continue searching while the *Main Agent* reasons and to deliver selected original evidence, rather than entire files, to the *Main Agent*'s context. Together, these designs broaden file inspection without repeatedly requiring a new *Main Agent* decision or loading the full contents of every inspected file. Across Qwen3.5-9B and Qwen3.6-27B on LoCoMo, HotpotQA, and 2WikiMultiHopQA, AFS achieves the highest mean F1 score while admitting 77–88% less unique source text into the Main Agent's context than Dynamic RAG. On 2WikiMultiHopQA with the 9B model, AFS improves F1 from 0.661 to 0.795 over Dynamic RAG while reducing mean online latency by 33.3%.



(a) AFS vs. current file systems. (b) F1 score vs. latency. (c) Source text visible to the *Main Agent*.

Figure 1: Comparison of AFS and conventional file systems. (a) File inspection and *Main Agent* reasoning process. (b) F1 score vs. online latency using different adaptive agentic file systems on 2WikiMultiHopQA with Qwen3.5-9B. (c) Total evidence tokens visible to the *Main Agent* per query.

1 INTRODUCTION

As LLMs become more capable, agents undertake increasingly complex tasks that require maintaining long contexts across extended interactions. File-intensive tasks further amplify this demand: relevant information may be distributed across many files that must be examined together, connected, and revisited as the task progresses. The challenge is therefore not only to accommodate more tokens, but to efficiently gather and use evidence from a growing workspace. RAG offers one way to access this information by ranking files according to semantic similarity and loading selected content into

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

the agent's context (Lewis et al., 2020). However, similarity-based selection does not ensure that the returned content contains precisely the evidence needed, and broader retrieval increases the amount of text the model must process (Liu et al., 2024). Iterative retrieval can refine this selection (Trivedi et al., 2023; Asai et al., 2024), but each round requires the reasoning model to interpret previous results and generate another request. File access thus remains tightly coupled to repeated model generations and a growing interaction history, increasing latency and context load.

Although agents can use command-line tools to navigate directories, search text, and read files, these interactions still follow workflows designed for human users (Anthropic, 2025; OpenAI, 2026; OpenCode, 2026). Agents translate information needs into successive operations such as `grep` and `glob`, then interpret the returned text before deciding what to inspect next, reproducing a human-style search-and-read loop. At the model level, however, processing file contents produces KV states that can be cached and reused. Conventional file interfaces do not expose files as independently reusable units of model computation: when the same content appears after different interaction histories, standard prefix caching cannot directly reuse its earlier KV states (Kwon et al., 2023; Zheng et al., 2024). This motivates us to rethink file access around reusable model computation, allowing files to be inspected efficiently without requiring the reasoning model to direct every search-and-read step.

In this paper, we introduce `AFS`, an agent-native file system that turns the file layer from a passive source of text into an active participant in evidence discovery (Figure 1). Rather than repeatedly folding file contents into the *Main Agent*'s interaction history, `AFS` treats files as independently inspectable units of reusable model computation. Three complementary designs let file inspection be prepared, batched, and continued alongside the *Main Agent*'s reasoning.

The workflow of `AFS` consists of offline preparation followed by concurrent file inspection and task reasoning. First, *offline preprocessing* prepares each file for repeated inspection. `AFS` assigns identifiers to original source spans and prepares routing representations and reusable per-file KV caches. Stable instructions and file content form a fixed input prefix before query-specific requests. Its KV states can therefore be reused across inspections while resident, independently of the *Main Agent*'s changing interaction history (Kwon et al., 2023; Zheng et al., 2024). *File Agents* select span identifiers, which the runtime resolves to original text, avoiding the additional autoregressive decoding and latency of reproducing evidence passages. Second, *parallel subagents* move local file inspection out of the *Main Agent*. *File Agents* identify potentially useful evidence within individual files, while the *Main Agent* handles the global task. Continuous batching executes these independent token sequences through shared model forward passes and admits new requests as others finish, improving inspection throughput without merging files into one large prompt. Third, *decoupled file inspection and reasoning* allows the two activities to progress concurrently. *File Agents* continue working while the *Main Agent* reasons, and the *Main Agent* can refine what evidence is needed without initiating every inspection. Selected source spans are restored as original records with provenance and delivered in updates. This separates how many files the system inspects from how much source text the *Main Agent* must consume.

We evaluate `AFS` with Qwen3.5-9B and Qwen3.6-27B on LoCoMo, HotpotQA, and 2WikiMultiHopQA (Maharana et al., 2024; Bai et al., 2024). In Table 1, `AFS` obtains the highest mean F1 in most settings. On LoCoMo, it improves F1 over Dynamic RAG by 0.090 and 0.118 with the 9B and 27B models, respectively. On 2WikiMultiHopQA, `AFS` raises F1 from 0.661 to 0.795 while reducing mean online latency from 27.81 to 18.56 seconds, a 33.3% reduction excluding per-file KV cache preparation. These joint gains suggest that broader evidence discovery need not prolong serial interactions when file inspection runs alongside reasoning.

Across all six settings, `AFS` admits 77–88% less unique original source text to the *Main Agent* than Dynamic RAG, illustrating that broader file inspection need not entail proportionally more evidence in the *Main Agent*'s context. Our analyses further examine file throughput, gold-evidence retention, and final evidence utilization to identify where the designs help and where evidence is still lost or misused.

Our contributions are threefold.

- We introduce an offline preparation scheme that equips files with addressable source spans, routing representations, and reusable per-file KV caches. This separates query-independent file computation from online requests, reducing repeated prefill and enabling exact source-text delivery without regenerating evidence passages.

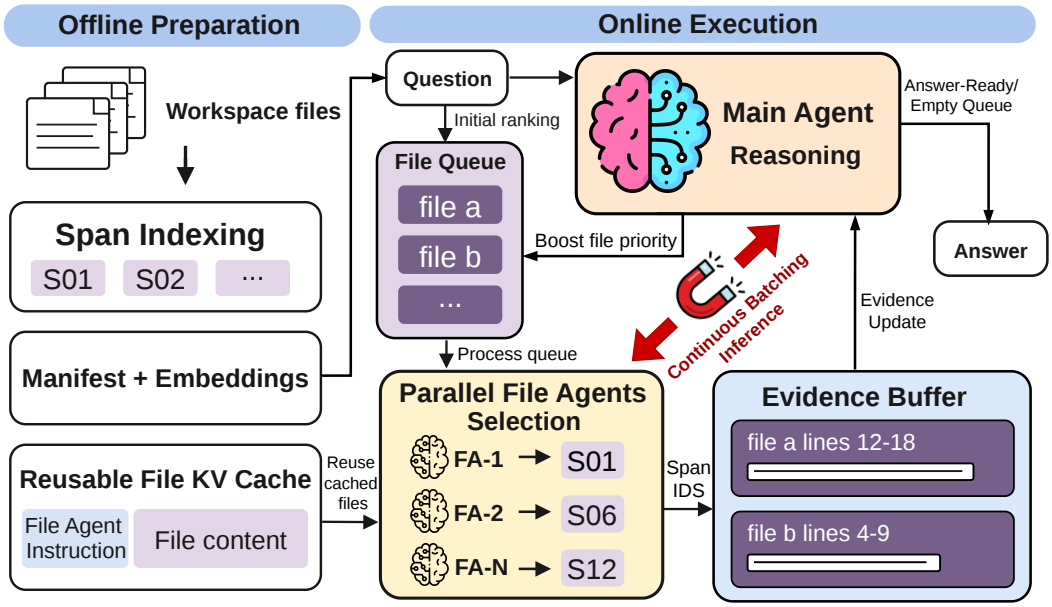


Figure 2: Overview of AFS. Offline preparation builds addressable spans, routing representations, and reusable per-file KV caches. Online, continuously batched *File Agent*$_\theta$ requests inspect queued files while *Main Agent*$_\theta$ reasons. Selected span IDs are resolved to original evidence and buffered for the next reasoning turn. *Main Agent*$_\theta$ can prioritize further inspection before answering.

- We develop an architecture that coordinates parallel *File Agents* with a single reasoning *Main Agent* on a shared, continuously batched model service. File inspection proceeds alongside reasoning, while selected original evidence is delivered through asynchronous updates. This broadens exploration without requiring a new Main Agent decision for every file or loading every inspected file into its context.
- We evaluate AFS across three benchmarks and two model scales, demonstrating favorable quality–latency trade-offs against the evaluated agentic retrieval and file-access baselines. Gold-evidence analyses and component ablations show how reusable computation, concurrent inspection, and selective evidence delivery enable efficient evidence acquisition while limiting the source text that the *Main Agent* must process.

2 Agent-Native File System

An agent-native file system should organize file access around model computation, rather than merely expose conventional file operations. AFS combines reusable per-file computation, parallel inspection, and asynchronous evidence delivery (Figure 2). Offline preparation makes each file independently inspectable through a cached input prefix. Online, *File Agents* locate useful records while a *Main Agent* reasons across the returned evidence. This division broadens inspection without requiring every file to enter the reasoning context or trigger a new interaction. We provide the algorithm in Appendix A.

Given a question q and files $\mathcal{D} = \{f_1, \ldots, f_N\}$, AFS uses one frozen backbone p_θ in two roles. We denote the *Main Agent* by *Main Agent*$_\theta$ and each *File Agent* by *File Agent*$_\theta$. Both use the same weights θ, with different instructions and independent contexts, rather than separately trained models. At reasoning turn t, H_t is the interaction history of *Main Agent*$_\theta$ and E_t is its admitted original evidence. Initially, $H_0 = [M, q]$ and $E_0 = \varnothing$, where M is the file manifest. The following stages connect reusable file representations to these evolving states.

2.1 Offline Preparation for Reusable Inspection

Source span indexing. For each file f, AFS constructs span identifiers $\mathcal{S}_f$ and a deterministic resolver ρ_f. Given selected IDs $S \subseteq \mathcal{S}_f$, $\rho_f(S)$ restores the original records with file and line locations and their source-local context. Thus, *File Agent*$_\theta$ generates compact locations rather than summaries or copied passages. Source addresses also support grouping and deduplication when evidence is delivered. Preparation produces manifest entries M and dense file embeddings $\mathbf{e}_f$ for online routing.

Reusable per-file KV caches. In conventional tool use, file content follows the question and an evolving interaction history. Its KV states depend on those preceding tokens, limiting reuse across different contexts. AFS instead places the fixed instructions I_{file} for *File Agent*$_\theta$ and the marked file before the question q and focused request r,

$$\begin{aligned} P_f = [I_{\text{file}}; \mathrm{Mark}(f)], \qquad K_f = \mathrm{KV}_\theta(P_f), \\ X_{f,q,r} = P_f \oplus [q; r]. \end{aligned} \tag{1}$$

Here $\oplus$ denotes concatenation, P_f is the fixed input prefix, and K_f is its reusable per-file KV cache. Changing q or r changes only the suffix. While the cache for the exact rendered P_f remains resident, inspection requires only suffix prefill and output decoding (Kwon et al., 2023; Zheng et al., 2024). This reuse occurs within *File Agent*$_\theta$ requests; cached states are never spliced into the reasoning history H_t.

2.2 Asynchronous File Inspection and Reasoning

Local selection, global reasoning. For each job $j = (f, r)$, *File Agent*$_\theta$ inspects $X_{f,q,r}$. The runtime validates selected span IDs and resolves them to original records,

$$\widehat{S}_j = \textit{File Agent}_\theta(X_{f,q,r}), \qquad R_j = \rho_f(\widehat{S}_j), \quad \widehat{S}_j \subseteq \mathcal{S}_f. \tag{2}$$

Malformed outputs are recorded as failures, not valid empty selections. We run *File Agent*$_\theta$ without thinking to select potentially useful local facts, including intermediate entities and updates that cannot answer q alone. In contrast, *Main Agent*$_\theta$ uses thinking to integrate records across files. Its action $a_t = \textit{Main Agent}_\theta(H_t)$ requests further inspections $\mathcal{J}_t = \{(f, r)\}$ or signals answer readiness. Local selection determines *what to expose*; global reasoning determines *how to combine it*.

Continuous batching. AFS couples agent coordination with model serving, rather than relying on message exchange alone. Online ranking initializes $\mathcal{Q}_0 = \mathrm{RankAll}(q, \{\mathbf{e}_f\})$, ordering inspection jobs instead of fixing a top-K reasoning context. Up to $B - 1$ *File Agents* draw from this queue alongside one *Main Agent*$_\theta$ generation. All requests share one model service, not separate replicas.

After input prefill, requests decode together while tokens within each sequence remain autoregressive. Let $\mathcal{B}_k$ be the decode requests scheduled at iteration k and $z_{\nu,k} \in \mathbb{R}^d$ a token's hidden row at a shared linear layer. The batched computation is

$$Z_k = \mathrm{Stack}_{\nu \in \mathcal{B}_k}(z_{\nu,k}), \qquad O_k = Z_k W, \quad W \in \mathbb{R}^{d \times d'}. \tag{3}$$

Rows share the weights W, but attention retains each request's own history and KV state. One forward pass can therefore advance *Main Agent*$_\theta$ and several *File Agent*$_\theta$ requests without merging their contexts. Since low-concurrency decoding is often memory-bandwidth limited, batching amortizes weight traffic across token rows (Kwon et al., 2023). It improves aggregate throughput, not necessarily each request's decoding speed.

Continuous batching replaces completed requests between iterations (Yu et al., 2022). Short span-selection requests finish and make room for new inspections while the longer reasoning request continues. Reusable per-file KV caches reduce the prefill needed for these replacements. Workers refill independently, without an all-worker barrier. The bound B controls application concurrency, not the engine's instantaneous batch size; actual admission also depends on token and KV cache capacity. Latency gains depend on useful evidence arriving promptly despite serving contention.

Asynchronous evidence updates. While *Main Agent*$_\theta$ reasons, *File Agents* continue inspecting queued files and buffer the selected original records. Requests in $\mathcal{J}_t$ can prioritize focused inspections, but workers need no new reasoning action to take the next background job. Let ΔE_{t+1} contain newly available records not already in E_t. If acquisition continues, these records enter the next reasoning turn after the current generation finishes,

$$\begin{aligned} E_{t+1} &= E_t \cup \Delta E_{t+1}, \\ H_{t+1} &= H_t \oplus a_t \oplus \mathrm{Update}(\Delta E_{t+1}). \end{aligned} \tag{4}$$

Here Update groups records by source. Several completed inspections can feed one update; empty or duplicate results do not trigger another generation. When inspection is complete or *Main Agent*$_\theta$ decides to stop acquisition, it reasons over the accumulated evidence to produce the final answer. Appendix Algorithm 1 summarizes preparation and the two concurrent online processes.

3 EXPERIMENTS

3.1 EXPERIMENTAL SETUP

Benchmarks and backbones. We evaluate Qwen3.5-9B[1] and Qwen3.6-27B[2] on three benchmarks. For efficient evaluation, we use a fixed 20% question subset comprising 397 questions of LoCoMo (Maharana et al., 2024), retaining each question's associated conversation corpus. For 2WikiMultiHopQA and HotpotQA (Ho et al., 2020; Yang et al., 2018), we use the LongBench versions (Bai et al., 2024), each containing 200 questions with supplied multi-document contexts. All methods receive the same questions and source corpora within each benchmark.

Hardware and serving. Each language-model instance runs on one NVIDIA H200 GPU with vLLM (Kwon et al., 2023) and tensor parallelism of one. `AFS` and its baselines share the same endpoint at each model scale. A shared Jina-v5-small retrieval encoder[3] computes query embeddings online; corpus embeddings are precomputed. The language-model instances enable prefix caching and priority scheduling with a 262,144-token context limit. `AFS` permits one *Main Agent*$_\theta$ request and up to seven *File Agent*$_\theta$ requests, with thinking enabled only for *Main Agent*$_\theta$. Archived primary-run settings specify an output cap of 16,384 tokens, temperature 1.0, top-$p = 0.95$, and top-$k = 20$.

Baselines. Our static baselines are One-shot RAG ($K = 1, 4$), which retrieves complete files once using the original question. The agentic baselines are Sequential Read, which opens one file per turn from a blind index; Dynamic RAG, which starts with four retrieved files and can reformulate queries; and Grep-based Harness, which exposes `glob`, `grep`, and `read` with multiple tool calls per turn. All methods acquire evidence before a separate final answer decode. Prompts can be found in Appendix B.

Metrics and comparison scope. We report official F1, mean online end-to-end latency (E2E), and unique original evidence tokens visible to the *Main Agent*. Online E2E covers evidence acquisition and final answer decoding from query arrival to the answer. We evaluate each setting using three different random seeds and report the mean and standard deviation across the three runs.

3.2 MAIN RESULTS

Static vs. agentic. Table 1 shows that adaptive agentic file access often improves answer quality over static retrieval, but incurs additional latency and evidence cost. Dynamic RAG outperforms One-shot RAG ($K = 4$) in five of six settings, while using more evidence tokens and taking longer in all settings. For example, on 2WikiMultiHopQA with Qwen3.5-9B, F1 increases from 0.52 to 0.66, but E2E rises from 22.50 to 27.81s and evidence grows from 3.39k to 4.01k tokens per query. Grep-based Harness admits less evidence than static $K = 4$ on both multi-hop benchmarks, but still takes longer. This contrast suggests that context size alone does not explain latency. Even selective tools leave successive search-and-read decisions on the model's serial execution path. *Thus, adaptive access can recover evidence missed by one-shot retrieval, yet does not by itself make evidence acquisition efficient.*

`AFS` improves the quality–latency trade-off. `AFS` improves this trade-off, obtaining the highest mean F1. On LoCoMo, Qwen3.5-9B improves F1 by 0.090 over Dynamic RAG and 0.086 over Grep-based Harness, with 15.7% and 5.2% lower reported E2E, respectively. Its evidence footprint is just 0.85k tokens per query, compared with 4.89k and 6.07k. On 2WikiMultiHopQA, the same model gains 0.134 F1 over Dynamic RAG while reducing E2E by 33.3%. At Qwen3.6-27B, `AFS` gains 0.119 F1 on LoCoMo and 0.022 on 2WikiMultiHopQA over Dynamic RAG, with 1.5% and 6.1% lower E2E. HotpotQA is essentially tied at 0.709 F1, with 2.2% lower E2E. Across all six

[1] `https://huggingface.co/Qwen/Qwen3.5-9B`
[2] `https://huggingface.co/Qwen/Qwen3.6-27B`
[3] `https://huggingface.co/jinaai/jina-embeddings-v5-text-small-retrieval`

settings, AFS admits 77–88% fewer evidence tokens than Dynamic RAG. The gains are consistent with AFS's separation of parallel file inspection from global reasoning, which can broaden discovery without requiring the *Main Agent* to read every inspected file. The analyses and matched-file ablation below examine how fewer serial interactions and selective original-record delivery contribute to this quality–latency trade-off.

Table 1: Results across three benchmarks and two model scales. Each setting has three runs, with standard errors reported. E2E is online latency (s). Evidence is unique source text visible to the *Main Agent* (k tokens/query). Higher F1 and lower E2E are better.

	LoCoMo			2WikiMultiHopQA			HotpotQA		
Method	F1-score	E2E (s)	Evidence (k tokens)	F1-score	E2E (s)	Evidence (k tokens)	F1-score	E2E (s)	Evidence (k tokens)
Qwen3.5-9B									
Static retrieval									
One-shot RAG ($K = 1$)	$0.30_{\pm 0.0072}$	$12.02_{\pm 0.02}$	$1.05_{\pm 0.000}$	$0.10_{\pm 0.0003}$	$18.75_{\pm 0.27}$	$0.68_{\pm 0.000}$	$0.30_{\pm 0.0003}$	$20.13_{\pm 0.40}$	$1.76_{\pm 0.000}$
One-shot RAG ($K = 4$)	$0.30_{\pm 0.0001}$	$15.33_{\pm 0.32}$	$4.16_{\pm 0.000}$	$0.52_{\pm 0.0016}$	$22.50_{\pm 0.26}$	$3.39_{\pm 0.000}$	$0.52_{\pm 0.0008}$	$22.45_{\pm 0.10}$	$7.84_{\pm 0.000}$
Agentic baselines									
Sequential Read	$0.29_{\pm 0.0037}$	$25.27_{\pm 0.37}$	$6.70_{\pm 0.148}$	$0.59_{\pm 0.0079}$	$28.24_{\pm 2.98}$	$5.51_{\pm 0.123}$	$0.54_{\pm 0.0125}$	$29.47_{\pm 2.27}$	$9.75_{\pm 0.278}$
Dynamic RAG	$0.28_{\pm 0.0003}$	$22.04_{\pm 0.14}$	$4.89_{\pm 0.017}$	$0.66_{\pm 0.0031}$	$27.81_{\pm 0.42}$	$4.01_{\pm 0.018}$	$0.60_{\pm 0.0023}$	$37.35_{\pm 1.53}$	$8.73_{\pm 0.117}$
Grep-based Harness	$0.28_{\pm 0.0058}$	$19.60_{\pm 1.27}$	$6.07_{\pm 0.298}$	$0.52_{\pm 0.0118}$	$25.57_{\pm 0.96}$	$1.72_{\pm 0.075}$	$0.53_{\pm 0.0085}$	$23.88_{\pm 1.54}$	$3.60_{\pm 0.094}$
AFS	$0.36_{\pm 0.0081}$	$18.58_{\pm 0.51}$	$0.85_{\pm 0.166}$	$0.80_{\pm 0.0113}$	$18.56_{\pm 0.25}$	$0.91_{\pm 0.031}$	$0.63_{\pm 0.0291}$	$23.83_{\pm 0.37}$	$1.56_{\pm 0.201}$
Qwen3.6-27B									
Static retrieval									
One-shot RAG ($K = 1$)	$0.31_{\pm 0.0001}$	$15.93_{\pm 0.15}$	$1.05_{\pm 0.000}$	$0.27_{\pm 0.0006}$	$22.53_{\pm 0.61}$	$0.68_{\pm 0.000}$	$0.45_{\pm 0.0064}$	$19.50_{\pm 0.01}$	$1.76_{\pm 0.000}$
One-shot RAG ($K = 4$)	$0.33_{\pm 0.0002}$	$17.09_{\pm 0.01}$	$4.17_{\pm 0.000}$	$0.70_{\pm 0.0040}$	$15.57_{\pm 0.01}$	$3.39_{\pm 0.000}$	$0.68_{\pm 0.0018}$	$15.20_{\pm 0.18}$	$7.84_{\pm 0.000}$
Agentic baselines									
Sequential Read	$0.29_{\pm 0.0040}$	$72.06_{\pm 2.51}$	$5.99_{\pm 0.102}$	$0.69_{\pm 0.0030}$	$50.54_{\pm 1.53}$	$4.54_{\pm 0.124}$	$0.67_{\pm 0.0085}$	$49.37_{\pm 0.04}$	$8.32_{\pm 0.088}$
Dynamic RAG	$0.33_{\pm 0.0019}$	$36.04_{\pm 0.07}$	$4.52_{\pm 0.026}$	$0.80_{\pm 0.0053}$	$27.74_{\pm 0.36}$	$3.98_{\pm 0.008}$	$0.71_{\pm 0.0023}$	$31.67_{\pm 0.25}$	$8.51_{\pm 0.013}$
Grep-based Harness	$0.28_{\pm 0.0229}$	$36.39_{\pm 3.84}$	$6.53_{\pm 0.016}$	$0.72_{\pm 0.0008}$	$28.72_{\pm 1.74}$	$1.86_{\pm 0.026}$	$0.69_{\pm 0.0026}$	$33.19_{\pm 2.40}$	$3.62_{\pm 0.107}$
AFS	$0.45_{\pm 0.0025}$	$35.49_{\pm 0.43}$	$0.61_{\pm 0.043}$	$0.82_{\pm 0.0008}$	$26.05_{\pm 0.52}$	$0.84_{\pm 0.074}$	$0.71_{\pm 0.0053}$	$30.96_{\pm 2.38}$	$1.05_{\pm 0.114}$

3.3 Observation and Analysis

We analyze the 397-question LoCoMo evaluation subset with Qwen3.5-9B to examine how file exploration translates into input to the *Main Agent* and visible evidence.

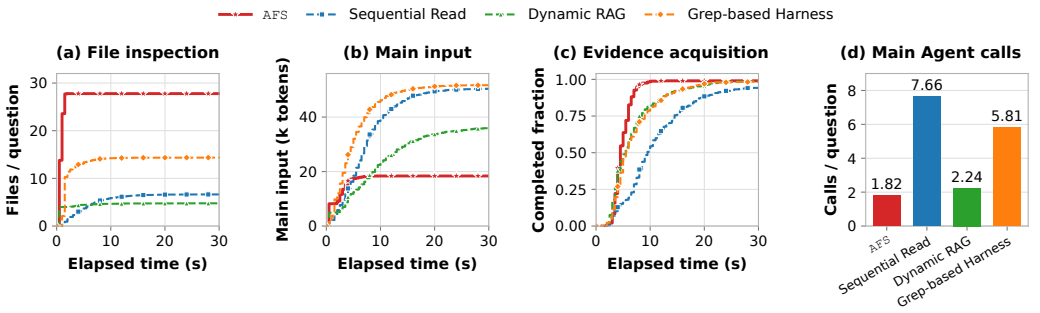


Figure 3: Agentic file access on LoCoMo with Qwen3.5-9B. (a) Distinct files explored. (b) Cumulative *Main Agent* input tokens, including repeated prompts and final decoding. (c) Fraction of queries completing evidence acquisition. (d) Mean *Main Agent* calls during evidence acquisition. All panels retain the same 397 questions, including failures. Time is measured from each query's start.

Broader exploration with fewer serial *Main Agent* decisions. AFS explores 27.78 files per question in 5.55s, compared with 4.77 files in 6.79s for Dynamic RAG, 14.37 in 6.97s for Grep-based Harness, and 6.64 in 10.26s for Sequential Read (Figure 3a and Figure 3c). Yet it uses only 1.82 acquisition *Main Agent* calls, versus 2.24, 5.81, and 7.66, respectively (Figure 3d). Its shorter acquisition therefore accompanies broader exploration, not a narrower search. This pattern is

consistent with the asynchronous design in Section 2.2. *File Agents* continue inspecting queued files during *Main Agent* reasoning, and multiple completions can feed one *Main Agent* update.

More file inspection need not mean more *Main Agent* input. AFS submits 18.38k cumulative input tokens to the *Main Agent*, compared with 36.71k for Dynamic RAG, 50.55k for Sequential Read, and 52.01k for Grep-based Harness (Figure 3b). Unlike the unique evidence measure in Table 1, these totals include repeated prompts and final decoding. Fewer *Main Agent* calls limit repeated history, while source-addressed selection avoids loading every inspected file in full. This supports treating inspection breadth and reasoning input as separate design choices.

Selective delivery can retain more useful evidence than full-file retrieval. For evidence coverage, we use the 308 answerable questions with gold annotations from this subset and compare all methods on the same questions (Table 2). We measure gold visibility (GV) as the mean fraction of labeled evidence intervals exposed to the *Main Agent*,

$$\mathrm{GV} = \frac{1}{|\mathcal{Q}_g|} \sum_{q \in \mathcal{Q}_g} \frac{|G_q^{\mathrm{vis}}|}{|G_q|}. \tag{5}$$

Here $\mathcal{Q}_g$ is the annotated question set, G_q contains the gold source intervals for question q, and $G_q^{\mathrm{vis}} \subseteq G_q$ contains intervals hit by original evidence actually submitted to the *Main Agent*.

A hit requires the same file and line ranges that overlap. GV averages per-question coverage, not pooled evidence counts or the fraction of questions with all gold visible. It measures annotation hits, not the completeness of an answer-bearing relation. AFS reaches 0.871 GV and 0.26 F1, versus 0.727 and 0.20 for One-shot RAG ($K = 4$), and 0.788 and 0.21 for Dynamic RAG. Returning complete retrieved files does not guarantee coverage if the relevant files are never retrieved. Grep-based Harness has slightly higher GV than AFS, but lower F1 (0.24 versus 0.26). Visibility alone thus does not explain the quality ordering. Source context and the *Main Agent*'s interpretation remain important. AFS's combination of wider exploration and local selection retains more labeled evidence here, despite not delivering whole files. Together, these results motivate both source-preserving selection and reliable integration of visible records, rather than maximizing access or coverage alone.

Table 2: Gold visibility (GV) and F1 score with Qwen3.5-9B. Both metrics use the same 308 answerable LoCoMo questions.

Method	GV	F1
RAG ($K = 1$)	0.457	0.19
RAG ($K = 4$)	0.727	0.20
Sequential Read	0.409	0.15
Dynamic RAG	0.788	0.21
Grep-based Harness	0.881	0.24
AFS	0.871	0.26

4 Ablations

We use fixed samples from LoCoMo with Qwen3.5-9B to examine the three designs introduced in Section 2, testing inspection startup cost, concurrent *File Agent* execution, and evidence selection. Each comparison uses the same samples across its settings. *File Agent* thinking results are in Appendix C.

Reusable per-file KV caches reduce inspection startup cost. We select 260 distinct file inputs from LoCoMo and compare their inspection startup with and without prebuilt per-file KV caches on a fresh service. Both conditions generate exactly one output token per request, isolating startup from variable-length evidence generation.

Prebuilt caches reduce mean time to first token (TTFT) from 919.7 to 297.7ms, a 68% reduction (Table 3). Processing the complete sample takes 12.38s rather than 35.54s, yielding 2.87× higher request throughput. This supports preparing file computation independently of the *Main Agent*'s changing history, so subsequent inspections can reuse it rather than process the entire file again.

Table 3: Effect of reusable per-file KV caches on inspection startup. Both conditions use the same 260 LoCoMo file inputs and generate one output token per request.

Metric	Without prebuilt KV cache	With prebuilt KV cache
TTFT (ms)	919.7	297.7
Workload (s)	35.54	12.38
Requests/s	7.32	21.00

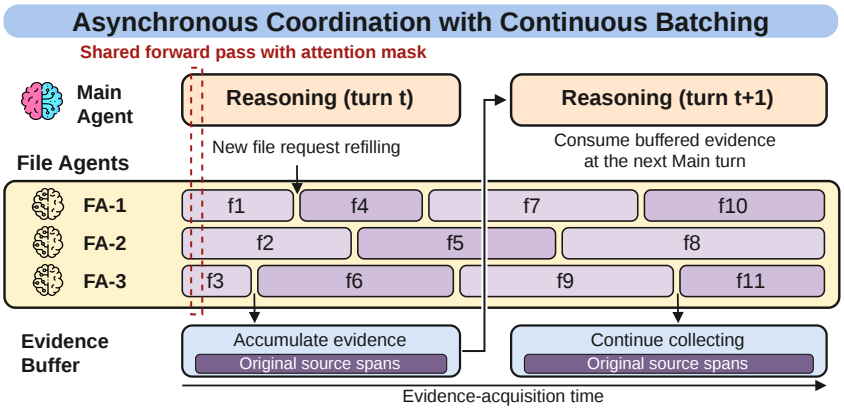


Figure 4: Asynchronous coordination with continuous batching. Original spans are buffered for the next *Main Agent* turn. The serving step is illustrative, not a measured batch trace; requests retain separate contexts.

Asynchronous coordination overlaps work, while continuous batching increases throughput. We select a fixed sample of 100 questions from the LoCoMo evaluation subset to study how *File Agent* inspection overlaps *Main Agent* reasoning and how concurrency affects inspection throughput. These mechanisms address different costs. Asynchronous coordination allows *File Agent* inspection to progress during *Main Agent* reasoning, while continuous batching increases the rate of processing independent inspections. Figure 4 illustrates how short *File Agent* requests replenish the batch and buffer original evidence for the next *Main Agent* turn.

In the AFS execution traces, *File Agent* inspection is active for 4.19s per question on average, of which 4.13s overlaps *Main Agent* execution. Overall, 98.5% of wall time with active *File Agents* is overlapped, leaving only 64ms per question when *File Agents* are active without the *Main Agent*. Thus most inspection proceeds alongside reasoning rather than in a separate waiting phase.

Table 4: *File Agent* concurrency at fixed work. Results average three cycles over 2,839 file-query pairs per cycle. We report p50 TTFT and decode latency.

File Agent count	1	3	7
Calls/s	2.64	4.27	4.31
Output tokens/s	23.2	37.7	38.0
TTFT (ms)	118	200	367
Decode (ms)	109	180	525

To isolate *File Agent* throughput, we collect the 2,839 file-query pairs initiated by the control on the same 100-question sample, deduplicating repeated requests within each question. We replay this fixed workload at *File Agent* concurrency $C \in \{1, 3, 7\}$ over three order-rotated cycles, holding service capacity, prompts, and stopping conditions fixed. The *Main Agent* is inactive in this comparison, and C limits in-flight requests rather than instantaneous engine batch size. Increasing C from one to three raises *File Agent* calls/s from 2.64 to 4.27 and output tokens/s from 23.2 to 37.7, approximately a 1.62× gain (Table 4). Increasing C to seven yields little further throughput but raises median decode latency from 180ms to 525ms and TTFT from 200ms to 367ms. The benefit is therefore efficient concurrent inspection, not maximizing concurrency.

Selection improves evidence use, not just input compactness. On the same sample of 100 LoCoMo questions, we compare evidence selection while fixing each question's inspected files, focused requests, and final decoder. This separates the effect of evidence selection from file discovery. *Full-file loading* passes the complete files, while *AFS selection* delivers original records selected by *File Agents* with local context expansion. Recall here pools annotated gold units, rather than averaging question-level coverage as in GV.

Table 5: Comparison of evidence selection with a fixed file set on 100 LoCoMo questions. Tokens/query and F1 use all questions; recall pools annotated gold units.

Selection	Tokens/q	Recall	F1
Full file	27,690	1.000	0.323
AFS	892	0.917	0.411

AFS retains 0.917 gold recall with 892 evidence tokens per query, versus 1.0 recall and 27,690 tokens for full-file loading (Table 5). Despite slightly lower recall, F1 rises from 0.323 to 0.411

while using only 3.2% of the source-text input. With file discovery held fixed, this gain points to the importance of what reaches the *Main Agent*, not simply how many files are found. Both answerable and unanswerable subsets improve, so the benefit is not limited to abstention.

5 Related Work

Retrieval and File Access for Agents. Retrieval-Augmented Generation (RAG) conditions generation on retrieved documents (Lewis et al., 2020). Subsequent approaches make access adaptive. IRCoT interleaves retrieval with intermediate reasoning (Trivedi et al., 2023), while Self-RAG learns when to retrieve and how to assess retrieved passages and generated responses (Asai et al., 2024). ReadAgent combines compressed gist memories with on-demand access to original passages (Lee et al., 2024). For coding tasks, SWE-agent designs an agent-computer interface for repository navigation, file editing, and program execution (Yang et al., 2024). Recent comparisons further show that the performance of grep and vector retrieval depends on the surrounding harness and tool-result presentation (Sen et al., 2026). More directly related to file access, LSFS provides prompt-driven semantic file operations (Shi et al., 2025), and CodeGrep trains a retrieval agent to locate candidate files for a downstream coding agent (Chen et al., 2026). These works improve how agents locate and revisit external information. `AFS` shares the goal of delegated access but focuses on continuous file-local inspection. *File Agents* repeatedly select original records in parallel while the *Main Agent* reasons, allowing inspection to proceed without a new *Main Agent* request for every file.

Efficient Inference for Agents. Efficient LLM serving exploits batching and computation reuse across requests. Orca schedules requests at the iteration level so that new work can join a running batch (Yu et al., 2022). vLLM reduces KV-cache fragmentation and duplication through PagedAttention (Kwon et al., 2023), while SGLang supports structured model programs and reuses KV states for shared input prefixes through RadixAttention (Zheng et al., 2024). More recently, KVFlow uses workflow structure to guide cache eviction and prefetching (Pan et al., 2025), and Continuum retains KV states across tool-call gaps through a time-to-live policy (Li et al., 2025). These systems improve how model requests are executed and cached. `AFS` builds on these serving mechanisms by organizing file access to exploit them. Reusable per-file KV caches avoid repeated file prefill independently of the *Main Agent*'s changing history, short *File Agent* requests replenish the shared batch during *Main Agent* reasoning, and span selection restricts the original text transferred to the *Main Agent*. This co-design connects inference scheduling to evidence discovery and delivery without additional model training.

6 Conclusion

We introduce `AFS`, an agent-native file system that co-designs file access with modern LLM serving. Reusable per-file KV caches, continuously batched *File Agents*, and asynchronous evidence delivery allow broad file inspection without proportional growth in the *Main Agent*'s context or serial interactions. Experiments with Qwen3.5-9B and Qwen3.6-27B on three benchmarks demonstrate favorable quality–latency trade-offs over strong retrieval and agentic baselines without model training. Our analyses show how reusable computation and overlapping inspection support efficient evidence acquisition, while selective transfer reduces the evidence processed by the *Main Agent*. Together, these findings motivate agent optimization beyond models and tools in isolation, toward joint design of the file layer and model execution.

Limitations and Future Work. Our evaluation focuses on question answering over static text corpora with two backbones from the same model family. Generalization to evolving coding workspaces remains untested. Reusable per-file KV caches require offline preparation and serving memory, whose costs are not fully reflected in online latency. Frequent file updates or limited cache capacity may therefore reduce the benefits of reuse. Moreover, local span selection can omit evidence needed to connect facts across files, and high gold visibility alone does not ensure correct answers. Future work should extend `AFS` to changing workspaces with update-aware cache management, and improve evidence preservation and coordination under constrained serving resources.

AI USE STATEMENT

In preparing this submission, we used LLMs as general-purpose writing assistants and coding agents to implement the proposed method and run experiments. The human authors were responsible for the entire research process, including initial ideation, experimental design, and data analysis. For the experiments, the authors first wrote down the methodological details and experimental plan, then instructed coding agents to implement the proposed method and run the experiments. During writing, the authors drafted each section, including all core ideas and technical details, and used LLMs to improve clarity, coherence, and grammar. All scientific claims and findings presented in this paper originate from the authors, who take full responsibility for the final content.

ETHICS STATEMENT

Our work adheres to the ICLR Code of Ethics. Deployments should restrict access to authorized files, protect sensitive information in caches and logs, and treat file contents as untrusted input that may contain prompt injection attempts. Any release or reuse of derived artifacts should comply with the source data's licenses and applicable privacy requirements.

REPRODUCIBILITY STATEMENT

We are committed to ensuring the reproducibility of our research. To this end, we will publicly release all code, data, and supplementary materials upon publication.

REFERENCES

Anthropic. Claude code: An agentic coding tool. `https://claude.com/product/claude-code`, 2025.

Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations*, 2024.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. LongBench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 3119–3137, 2024.

Wuya Chen, Yihao Yang, Yang Cao, and Yue Lin. CodeGrep: An RL-trained retrieval agent for LLM coding agents. *arXiv preprint arXiv:2608.05886*, 2026.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6609–6625, 2020.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th symposium on operating systems principles*, pp. 611–626, 2023.

Kuang-Huei Lee, Xinyun Chen, Hiroki Furuta, John Canny, and Ian Fischer. A human-inspired reading agent with gist memory of very long contexts. *arXiv preprint arXiv:2402.09727*, 2024.

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. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, 2020.

Hanchen Li, Runyuan He, Qiuyang Mang, Qizheng Zhang, Huanzhi Mao, Xiaokun Chen, Hangrui Zhou, Huanchen Zhang, Alvin Cheung, Joseph Gonzalez, and Ion Stoica. Continuum: Efficient and robust multi-turn LLM agent scheduling with KV cache time-to-live. *arXiv preprint arXiv:2511.02230*, 2025.

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

Adyasha Maharana, Dong-Ho Lee, Sergey Tulyakov, Mohit Bansal, Francesco Barbieri, and Yuwei Fang. Evaluating very long-term conversational memory of LLM agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pp. 13851–13870, 2024.

OpenAI. Harness engineering: Leveraging codex in an agent-first world. `https://openai.com/index/harness-engineering/`, 2026.

OpenCode. Opencode tools. `https://opencode.ai/v2/docs/tools/`, 2026.

Zaifeng Pan, Ajjkumar Patel, Zhengding Hu, Yipeng Shen, Yue Guan, Wan-Lu Li, Lianhui Qin, Yida Wang, and Yufei Ding. KVFlow: Efficient prefix caching for accelerating LLM-based multi-agent workflows. *arXiv preprint arXiv:2507.07400*, 2025.

Sahil Sen, Akhil Kasturi, Elias Lumer, Anmol Gulati, and Vamse Kumar Subbiah. Is grep all you need? how agent harnesses reshape agentic search. *arXiv preprint arXiv:2605.15184*, 2026.

Zeru Shi, Kai Mei, Mingyu Jin, Yongye Su, Chaoji Zuo, Wenyue Hua, Wujiang Xu, Yujie Ren, Zirui Liu, Mengnan Du, Dong Deng, and Yongfeng Zhang. From commands to prompts: LLM-based semantic file system for AIOS. In *International Conference on Learning Representations*, 2025.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 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*, pp. 10014–10037, 2023.

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

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 2369–2380, 2018.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538. USENIX Association, 2022.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, 2024.

A AFS ALGORITHM

Algorithm 1 summarizes the three designs introduced in Section 2 and Figure 2. Prebuilt per-file KV caches support reuse; parallel *File Agents* select original evidence; the *Main Agent* reasons while inspection continues. The two online blocks run concurrently on the same continuously batched model service, with thinking enabled for the *Main Agent* and disabled for *File Agents*.

Algorithm 1 AFS with reusable files and concurrent evidence discovery

Require: Files $\mathcal{D}$, question q, concurrency B
Ensure: Answer y
Offline preparation
Index source spans $(\mathcal{S}_f, \rho_f)$; build manifest M and file embeddings $\mathbf{e}_f$
Prepare fixed file prefixes P_f and reusable per-file KV caches K_f
Online initialization
$\mathcal{Q} \leftarrow \mathrm{RankAll}(q, \{\mathbf{e}_f\})$
$H_0 \leftarrow [M, q]$; $E_0 \leftarrow \varnothing$; $U \leftarrow \varnothing$; $t \leftarrow 0$
File Agent inspection by up to $B-1$ workers, alongside the *Main Agent*
$j = (f, r) \leftarrow$ next eligible request in $\mathcal{Q}$
$\widehat{S}_j \leftarrow$ *File Agent*$_\theta(P_f \oplus [q; r])$
Buffer validated, newly selected original records $\rho_f(\widehat{S}_j)$ in U
Main Agent reasoning, concurrent with *File Agent* inspection
while evidence acquisition continues **do**
  $a_t \leftarrow$ *Main Agent*$_\theta(H_t)$
  if the *Main Agent* declares answer readiness **then**
    break
  end if
  Add or prioritize the *Main Agent*'s requested inspections in $\mathcal{Q}$
  Wait for the next evidence or search-status update u while acquisition continues
  $\Delta E_{t+1} \leftarrow$ new source records in u
  $E_{t+1} \leftarrow E_t \cup \Delta E_{t+1}$; $H_{t+1} \leftarrow H_t \oplus a_t \oplus u$
  $t \leftarrow t + 1$
end while
Final answering
$E_T \leftarrow E_t$; $y \leftarrow \textsc{FinalAnswer}(q, E_T)$ ▷ fresh context, no tools
return y

Workers refill independently of the *Main Agent*. Before a new *Main Agent* turn, one update collects buffered records, removes source duplicates, and groups the originals by file, as in Eq. 4. The *Main Agent*'s input stays fixed during generation; individual empty or duplicate *File Agent* results do not trigger another turn. Identical file-request pairs reuse results, while new requests may revisit a file.

Acquisition ends on readiness, search exhaustion after any configured terminal notice, or its budget limit. Exhaustion includes running inspections and unfinished ranking, not just an empty queue. Final decoding uses the frozen admitted evidence E_T, not late *File Agent* completions. An acquisition candidate is additionally supplied only under the declared handoff in Appendix B.

B PROMPTS

We list the task instructions and message templates for each method. Braces denote runtime inputs. Tool schemas are supplied through the model's native tool interface. *Main Agent*$_\theta$ uses thinking, while *File Agent*$_\theta$ uses no-thinking in the main experiments.

B.1 SHARED TASK INSTRUCTIONS

Baseline system messages start with the task instruction for the corresponding benchmark. The placeholders below refer to these exact blocks.

Memory task instruction (LoCoMo)

```
You are helping a user answer a question from saved conversation
and memory records. Base the answer on information found in
those records. Pay close attention to who said or did something,
the relation being asked about, dates, later updates, negation,
and whether the question asks for one fact or a complete set. If
the available records do not support the answer, say "No
information available" rather than guessing.
```

Passage task instruction (HotpotQA and 2WikiMultiHopQA)

```
Answer the question using only the supplied passages. Keep
entities, relations, comparison direction, and multi-hop links
attached to the passage that states them. Give only the answer
and do not describe the retrieval process.
```

For memory questions, `{question_block}` contains `CURRENT DATE: {question_date}` when a date is provided, followed by `QUESTION: {question}`. For passage questions, it contains only `QUESTION: {question}`.

B.2 ONE-SHOT RAG

All retrieval depths use the same prompt. Only the complete files inserted into `{original_records}` change.

One-shot RAG on LoCoMo

```
[System]
{memory_task_instruction}

Write the final response after 'Answer:'. Give the answer
itself, without describing the retrieval process.

[User]
The following history records were retrieved as potentially
relevant to the question. Please answer using the relevant
information in them.

RETRIEVED HISTORY RECORDS:
{original_records}

{question_block}
Answer:
```

One-shot RAG on HotpotQA and 2WikiMultiHopQA

```
[System]
{passage_task_instruction}

[User]
The following are given passages.

{original_records}

Answer the question based on the given passages. Only give me
the answer and do not output any other words.

Question: {question}
Answer:
```

B.3 SEQUENTIAL READ

The file index contains opaque identifiers and routing-neutral metadata, without file summaries or answer evidence.

Sequential Read on LoCoMo

```
[System]
{memory_task_instruction}

You are given an index containing only opaque file IDs, dates,
and line counts. Use that metadata to decide which original file
to inspect. The index contains no answer evidence. Tool results
remain available in the conversation.

Choose the next action according to these conditions:
- If more information is needed, call `read_history_file` to
  inspect one original file.
- If the evidence is sufficient, call
  `finish_evidence_collection`. That tool only ends evidence
  collection; it does not contain the answer.

While evidence-collection tools are available, call exactly one
tool in each turn. After evidence collection finishes, you will
answer naturally in this same conversation without tools.

[User]
HISTORY FILE INDEX:
{file_index}

{question_block}
```

Sequential Read on HotpotQA and 2WikiMultiHopQA

```
[System]
{passage_task_instruction}

You are given an index containing only opaque source file IDs
and routing-neutral metadata. It contains no passage title,
summary, or answer evidence. Explore by choosing complete
original passages to inspect, and verify every answer fact in
their contents. Tool results remain available in the
conversation.

Choose the next action according to these conditions:
- If more information is needed, call `read_source_file` to
  inspect one original file.
- If the evidence is sufficient, call `answer`. This is the
```

Sequential Read on HotpotQA and 2WikiMultiHopQA (continued)

```
  ordinary search-or-answer decision; its payload is not scored
  or passed to the mandatory fresh final.

While acquisition tools are available, call exactly one tool in
each turn. After evidence collection finishes, a separate
clean-context call will answer from the original evidence you
acquired.

[User]
SOURCE PASSAGE FILE INDEX:
{file_index}

{question_block}
```

B.4 DYNAMIC RAG

Dynamic RAG on LoCoMo

```
[System]
{memory_task_instruction}

Some potentially relevant history files have already been
retrieved. Retrieved files remain available in the conversation.

Choose the next action according to these conditions:
- If more information is needed, call `search_history_files`
  with a focused query for the missing information.
- If the evidence is sufficient, call
  `finish_evidence_collection`. That tool only ends evidence
  collection; it does not contain the answer.

While evidence-collection tools are available, call exactly one
tool in each turn. After evidence collection finishes, you will
answer naturally in this same conversation without tools.

[User]
RETRIEVED HISTORY RECORDS:
{initial_retrieved_files}

{question_block}
```

Dynamic RAG on HotpotQA and 2WikiMultiHopQA

```
[System]
{passage_task_instruction}

Some potentially relevant source passages have already been
retrieved. Retrieved passages remain available in the
conversation.

Choose the next action according to these conditions:
- If more information is needed, call `search_source_files` with
  a focused query for the missing information.
- If the evidence is sufficient, call `answer`. This is the
  ordinary search-or-answer decision; its payload is not scored
  or passed to the mandatory fresh final.

While acquisition tools are available, call exactly one tool in
each turn. After evidence collection finishes, a separate
clean-context call will answer from the original evidence you
acquired.
```

Dynamic RAG on HotpotQA and 2WikiMultiHopQA (continued)

```
[User]
RETRIEVED SOURCE PASSAGES:
{initial_retrieved_files}

{question_block}
```

B.5 GREP-BASED HARNESS

Grep-based Harness on LoCoMo

```
[System]
{memory_task_instruction}

The history is stored in files. Use `glob` to discover paths,
`grep` to search original lines across files, and `read` to
inspect original file contents. Tool results remain available in
the conversation.

Choose the next action according to these conditions:
- If more information is needed, call one or more of `glob`,
  `grep`, and `read`.
- If the evidence is sufficient, call
  `finish_evidence_collection`. That tool only ends evidence
  collection; it does not contain the answer.

You may issue multiple filesystem tool calls in one turn. Never
combine `finish_evidence_collection` with another tool call.
After evidence collection finishes, answer naturally in this
same conversation without tools.

[User]
{question_block}
```

Grep-based Harness on HotpotQA and 2WikiMultiHopQA

```
[System]
{passage_task_instruction}

The source passages are stored in files. Use `glob` to discover
paths, `grep` to search original lines across files, and `read`
to inspect original file contents. Tool results remain available
in the conversation.

Choose the next action according to these conditions:
- If more information is needed, call one or more of `glob`,
  `grep`, and `read`.
- If the evidence is sufficient, call `answer`. This is the
  ordinary search-or-answer decision; its payload is not scored
  or passed to the mandatory fresh final.

You may issue multiple filesystem tool calls in one turn. Never
combine `answer` with another tool call. After evidence
collection finishes, a separate clean-context call will answer
from the original evidence you acquired.

[User]
{question_block}
```

B.6 AFS

***Main Agent*.** The system instructions below accompany the file index and question. Subsequent updates contain original evidence grouped by source and the status of inspection requests.

AFS *Main Agent* on LoCoMo

```
You will answer a question using saved conversation history.

You will receive an index of history files and passages
retrieved from those files. Use the index to find relevant
files, and base your answer on the retrieved history. Previously
returned passages remain available in this conversation.

At each step:
- If more information is needed, call request_files. Describe
  the facts you still need and what to look for in each file.
- If an earlier search did not find those facts, you may request
  the file again with a more specific search request, within the
  available search budget.
- If the available information is sufficient, combine the
  relevant facts and call answer with your response.
- If no further inspection is available and the history does not
  support an answer, call answer with "No information
  available".

An empty search result means no passages were selected for that
request. A failed search means the request could not be
completed.

Call one tool at a time.

When deciding whether the records support an answer:
- Keep people, actions, objects, recipients, source events, and
  time attached to the same event. Similar words or topics from
  different events cannot fill missing roles in a relationship.
- The relation may be expressed by a natural paraphrase; it need
  not repeat the question's exact words.
- Search further only when a missing role, conflicting event,
  time constraint, or coverage requirement could change the
  answer.

Once one provenance-consistent event or event set satisfies
every constraint requested by the question, call answer in this
turn. Do not continue searching for a verbatim restatement of an
already supported relation. In the answer field, return only the
requested fact or complete set, without retrieval narration or
unrelated alternatives.

When the question asks for an inference rather than a verbatim
fact, a unique provenance-consistent implication from a person's
repeated observations, routines, actions, or updates can support
the answer. Do not replace the requested inference with a
stricter requirement for an explicit address, label, or exact
restatement. Conflicting roles, events, or times still require
resolution before answering.

Keep all needed reasoning in your thinking. When you call
answer, put only the fact or complete set requested by the
question in the answer field, normally as a short phrase or
sentence. Omit evidence narration, citations, retrieval
commentary, unasked alternatives, and general background.
```

AFS *Main Agent* on HotpotQA and 2WikiMultiHopQA

```
You will answer a question using a collection of source
passages.

You will receive an index of passage files and original spans
returned from those files. The index is routing metadata; base
the answer only on original spans that have entered this
conversation. Previously returned spans remain available.

Choose exactly one action at each step:
- If more information is needed, call request_files. State the
  missing fact or link and what to look for in each requested
  file.
- If the available original spans are sufficient, combine the
  relevant facts and call answer.
- If all available file inspection is exhausted and the passages
  do not support an answer, call answer with "No information
  available".

For a multi-hop or comparison question, keep each entity,
relation, date, and comparison direction attached to its source
passage before joining the facts. Once the chain is complete,
answer immediately. Put only the requested answer in the answer
field; do not include retrieval narration, citations, or
unrelated details. Call one tool at a time.
```

***File Agent*.** Each *File Agent* receives one marked file, the question, and a focused request. Selected span identifiers are resolved to original text.

AFS *File Agent* on LoCoMo

```
I will give you a history file, a question, and a search
request. Please select passages containing information that may
help answer the question.

Use the question as context and the search request to guide what
to look for.

- If a passage provides relevant facts, select its span ID. It
  may provide only part of the information needed, rather than
  an answer to the whole question.
- If those facts need surrounding context to be understood,
  select the relevant context spans as well.
- If no passage may help, return an empty selection.

Each span is enclosed by BEGIN_SPAN and END_SPAN. Use the IDs on
these markers, not numbers appearing in the history.

Return only a JSON object containing "span_ids". For an empty
selection, return {"span_ids": []}. Do not write an answer or
repeat the passages.
```

AFS *File Agent* on HotpotQA

```
I will give you one source passage, a question, and a focused
request. Select every original span in this passage that may
provide a direct fact, a bridge fact, or comparison input needed
for the question.

- Use the focused request to guide the search and the complete
  question to preserve entity identity, relation direction, and
  multi-hop scope.
- A span may be useful even when it supplies only one link
  rather than the final answer.
- Include surrounding spans only when needed to understand the
  selected fact.
- Return an empty selection only when no span in this passage
  may help.

Each span is enclosed by BEGIN_SPAN and END_SPAN. Return only a
JSON object containing "span_ids" and use only those marker IDs.
Do not answer, summarize, or rewrite the source.
```

AFS *File Agent* for multi-hop passage selection

```
You are the high-recall passage reader in a multi-hop
question-answering system. I will give you one complete source
passage, the original question, and a focused request. Select
every original span in this passage that may establish any local
link needed to answer the question.

- This passage may contain only one hop, so it need not mention
  every entity in the original question or contain the final
  answer.
- Preserve entity-to-entity links and entity-to-attribute facts.
  For a comparison, preserve both the item-to-subject link and
  the compared attribute whenever present.
- Check the title and lead paragraph, including dates or
  attributes in parentheses.
- If useful links occur in distant parts of the same passage,
  select all corresponding spans. A selected relationship may
  make an earlier lead span relevant, and vice versa.
- Treat the focused request as a search priority, not an
  exclusion rule. If its wording is incomplete but the passage
  clearly supplies a needed hop, preserve that hop.
- Return an empty selection only when no span can support any
  link in the question's reasoning chain. Favor retaining a
  plausible local fact over dropping it.

Each span is enclosed by BEGIN_SPAN and END_SPAN. Return only a
JSON object containing "span_ids" and use only those marker IDs.
Do not answer the question, summarize, infer the final
cross-passage result, or rewrite the source.
```

AFS *File Agent* user message

```
FILE_ID: {file_id}
ORIGINAL SPANS:
{marked_original_spans}

USER QUESTION: {question}
FOCUSED REQUEST: {request}
```

Action-format instructions. The following supplementary instructions apply when native-action enforcement and compact JSON output are enabled, respectively.

AFS *Main Agent* native-action instruction

```
Every Main response must end with exactly one native tool call.
A response containing only free-form text is invalid. Choose the
action from the original-record state already visible in this
conversation.

HARD INITIAL ROUTING RULE: the first Main turn has no ORIGINAL
EVENT PACKET, so it is impossible to have enough evidence to
answer. The HISTORY FILE INDEX is routing metadata only. Do not
quote, paraphrase, compare, or reason from its summaries. Your
entire first-turn thinking must be one routing sentence of the
form "Inspect FILE_ID for MISSING_FACT." Then immediately call
request_files with exactly one request for that file. No other
first-turn action is valid.

- If some original records are visible but they are
  insufficient, use at most 80 thinking words to identify only
  the specific unresolved relation, then immediately call
  request_files. Do not restate the evidence already known.
- In request_files, requests must be a JSON array of request
  objects. Never quote or stringify that array, and never put
  escaped JSON inside a string.
- If the visible original records are sufficient, do not restate
  the question, search process, or evidence record by record. In
  at most 80 thinking words, check only the subject, requested
  relation, time scope, and set completeness, then immediately
  call answer. Do not continue free-form text after deciding
  either action. Keep the answer payload concise as already
  specified.
```

AFS *File Agent* JSON-output instruction

```
Before emitting any output, decide the complete list of selected
span IDs. Then output exactly one compact JSON object and stop:
{"span_ids":["ID",...]}. If no span may help, output
{"span_ids":[]}. Never revise or correct an emitted object.
Never output Markdown, commentary, an explanation, a repeated
passage, or text after the closing brace.
```

B.7 FINAL-ANSWER PROMPTS

The baselines use the One-shot RAG message templates with acquired original records in place of retrieved files. AFS uses the same memory template on LoCoMo. The quoted memory acquisition prompts retain their same-conversation wording; in the reported clean-context setting, the controller constructs a separate final call from the acquired evidence. The AFS passage-final instructions are given below.

AFS final answer on 2WikiMultiHopQA

```
[System]
Answer the question using only the supplied passages. Think
internally to resolve the exact named entities, relation chain,
comparison direction, and requested answer type. Passage
identifiers and provenance are reasoning aids only: never print
them in the final response. Once the requested answer value is
supported, finish the internal reasoning instead of reopening
the same conclusion unless another supplied passage explicitly
contradicts it. Output only the requested answer value.
```

AFS final answer on 2WikiMultiHopQA (continued)

```
[User]
The following are given passages.

{original_records}

Answer the question based on the given passages. Only give me
the answer and do not output any other words.

Question: {question}
Answer:

FINAL RESPONSE CONTRACT: infer the requested value type from the
complete relation phrase, not from the question word alone.
Resolve all required hops internally. Then write only the value
that fills that exact slot after 'Answer:'. Do not include
reasoning, citations, passage IDs, provenance, labels, or an
explanation.
```

On HotpotQA, AFS uses the passage-final template and appends the following block when a nonempty, non-refusal acquisition draft is available.

AFS final-answer user-message addition on HotpotQA

```
CANDIDATE ANSWER FROM EVIDENCE COLLECTION:
{candidate}

The candidate is not evidence. Verify it against the original
passages. If it is supported and already has the granularity
requested by the question, preserve it. If it is incomplete or
contradicted, correct it. Return only the requested answer.
```

C File Agent Thinking

On a fixed sample of 100 LoCoMo questions with Qwen3.5-9B, *File Agent* thinking slightly improves F1 from 0.415 to 0.423, while observed mean E2E rises from 17.59s to 95.32s (Table 6). The *Main Agent* always retains thinking. Shared-GPU load prevents isolating the timing effect, but these runs show a small quality gain accompanied by much higher latency.

Table 6: *File Agent* thinking in full-system AFS. The 512-token first-generation cap permits up to 256 additional tokens for span IDs if truncated. E2E is measured under shared-GPU load.

Metric	No thinking	Thinking (512)
Official F1	0.415	0.423
Mean E2E (s)	17.59	95.32

We separately compare 2K, 4K, and 8K budgets on 251 fixed file-query pairs (Table 7). Gold-unit recall of *File Agent* selections remains high at 0.930, 0.940, and 0.910, respectively, but does not improve monotonically. Longer *File Agent* thinking therefore does not necessarily yield more complete evidence, supporting lightweight *File Agent* selection with extended reasoning reserved for the *Main Agent*.

Table 7: Evidence coverage at different *File Agent* thinking budgets on 251 fixed file-query pairs. Coverage is gold-unit recall of *File Agent* selections.

Thinking budget	None	2K	4K	8K
Gold coverage	0.890	0.930	0.940	0.910