CLAW: A File-First Personal AI Operating System with Typed Agent Composition, Budget Governance, and Temporal Memory

Hamed Arab
Maker's AI Lab

June 2026

Abstract

Recent work argues that the filesystem itself can replace much of the orchestration code in agentic AI systems. The Interpretable Context Methodology (ICM) shows that numbered folders and plain markdown files can coordinate a single agent through a sequential, human-reviewed workflow without a multi-agent framework. The Open Knowledge Format (OKF) proposes a vendor-neutral markdown-plus-YAML standard so that knowledge authored by one producer can be consumed by any agent. Both formalise a pattern that practitioners had been building by hand in `CLAUDE.md` and `AGENTS.md` files. This paper reports on CLAW, a file-first personal AI operating system that has run continuously for over a month as the single operator's daily automation substrate across business and personal domains. CLAW shares the file-first foundation of ICM and OKF but extends it along five axes that sequential single-agent content pipelines do not require: intent routing before context loading, typed multi-agent composition with per-role tool budgets, a four-level cost-governance ladder, temporal memory with decay and full-text search, and event-driven proactivity gated by cheap guards. We describe the architecture, then report measured operating data: 94 skills and 15 typed agents addressed through one router, 5,183 scheduled-run records over 41 days, a 98 per cent clean-checkpoint rate across 50 skills, a durable-memory index that holds the always-resident knowledge surface to 19 per cent of the full memory store, and on-demand skill loading plus automated memory hygiene that keep the resident framing context roughly thirteen times smaller than a monolithic baseline. We position CLAW relative to ICM, OKF, and the multi-agent framework literature, and we make the system's skill library OKF-compatible to demonstrate interoperability. We are explicit about the limits: this is single-operator, single-model-family, observational evidence, not a controlled study.

1 Introduction

There are good agentic frameworks. CrewAI, LangChain, and AutoGen handle multi-step orchestration, memory, tool use, and error recovery in code. Van Clief and McDermott observe that for sequential workflows where a human reviews each step, that machinery can be more than the problem needs, and that folder structure plus markdown files can carry the same coordination [1]. Google's Open Knowledge Format makes a parallel argument at the knowledge layer: the unit that should be portable between agents is not a service but a format, a directory of markdown files with YAML frontmatter [2].

Both proposals describe, and partly standardise, a pattern that practitioners had already adopted in `CLAUDE.md` and `AGENTS.md` convention files. This paper reports on a system that took that pattern as a starting point and built outward from it into something closer to a personal operating system. CLAW (Claude-Local Agent Workbench) is the single author's daily automation substrate. It runs a jewellery e-commerce brand's content and SEO pipeline, a training academy's marketing and product work, personal finance ingestion, email triage across

three accounts, meal planning, and the maintenance of its own durable memory. It has done so continuously for over a month, mostly unattended, on a consumer Windows machine driving a subscription language model.

The central question of ICM is how filesystem-structured context affects a practitioner's ability to control a sequential workflow. CLAW raises a different question that ICM explicitly scopes out: what does the file-first approach look like when the workflow is not a single human-initiated sequence but a long-running, multi-domain, partly autonomous operation with dozens of concurrent concerns, real spending, and irreversible external actions. Answering it requires five capabilities that ICM's sequential-pipeline model does not address, and that the multi-agent frameworks address only in code.

Contributions. This paper makes four contributions.

1. A description of CLAW's architecture: a file-first system that keeps ICM's layered context hierarchy and OKF's format discipline, and adds intent routing, typed agent composition with per-role tool budgets, a cost-governance ladder, temporal memory with decay and full-text search, and guard-gated event-driven proactivity (Section 3).

2. An evaluation grounded entirely in measured operating data from a live deployment, including system scale, context-efficiency ratios, checkpoint reliability, autonomous run volume, human-in-the-loop resolution rates, and the safety surface (Section 4).

3. A demonstration that the same architecture supports de-personalised portability: MakerOS, a stripped copy of CLAW that non-technical users install and configure for their own context.

4. An OKF-compatibility result: we made all 94 skill definitions valid OKF documents and produced skill-library index files, so that any OKF-aware consumer can navigate the library without loading skill bodies (Section 3.10, Section 5).

We are candid throughout about what this evidence is and is not. It is the detailed record of one system used heavily by one operator, not a controlled comparison. Section 6 states the threats to validity in full.

2 Related Work

2.1 File-first orchestration

Van Clief and McDermott's Interpretable Context Methodology is the closest prior work [1]. ICM replaces framework orchestration with filesystem structure: numbered folders are stages, markdown `CONTEXT.md` files carry per-stage prompts, local scripts do the non-AI work, and one orchestrating agent reads the right files at each step. ICM defines a five-layer context hierarchy (identity, routing, stage contract, reference material, working artifacts) and reports that scoped per-stage loading keeps each step's context to 2,000 to 8,000 tokens against a monolithic 40,000-plus. ICM's scope is explicit: sequential, human-reviewed, repeatable workflows, single model family, observational evidence from a practitioner community. It states that real-time multi-agent collaboration, high concurrency, and automated mid-pipeline branching fall outside its design. CLAW adopts ICM's layered hierarchy wholesale and then builds the capabilities ICM scopes out.

2.2 Knowledge formats

The Open Knowledge Format is a vendor-neutral specification for the knowledge that agents consume: a directory of markdown files with YAML frontmatter, one mandatory field (`type`),

reserved `index.md` and `log.md` filenames, and cross-links as ordinary markdown links forming an implicit graph [2]. OKF formalises the "LLM-wiki" pattern and names `AGENTS.md` and `CLAUDE.md` conventions as its prior art. CLAW predates OKF but instantiates the same pattern; Section 3.10 reports making CLAW's skill library OKF-compatible.

2.3 Multi-agent frameworks

MetaGPT assigns software-engineering roles to multiple agents and coordinates them through code, evaluated on coding benchmarks [3]. AutoGen models applications as conversations between configurable agents and provides the message-passing infrastructure for dynamic multi-agent collaboration [4]. LangChain and CrewAI offer similar code-level abstractions. These systems excel where ICM and CLAW do not: tight-loop agent-to-agent dialogue and high concurrency. CLAW differs in target and in mechanism. Its target is one person's life and business automation rather than software production, and its coordination lives in files and a routing protocol rather than in a message bus. Where CLAW composes multiple agents, it does so as a shallow orchestrator-and-workers tree with file handoffs, not as a peer dialogue.

2.4 Agent memory

MemGPT frames the language model as an operating system that pages information between a fixed context window and external storage [5]. CLAW's memory pursues the same goal, surviving beyond a single context window, but with a deliberately simpler and human-legible mechanism: an always-loaded index, a recent-detail store, and a full-text-searchable archive of dated plain-markdown entries, trimmed on a cadence. Park et al.'s generative agents demonstrate memory streams with reflection in a simulated environment [6]; CLAW's reflection is the explicit promotion of daily-log observations into durable memory by a dedicated curator agent.

2.5 Context engineering

The efficiency argument for scoped context rests on Liu et al., who show that language models perform worse when relevant information is buried in long contexts [7]. Wei et al. show that decomposing a task into intermediate steps improves performance [8]. Karpathy's framing of "context engineering" over "prompt engineering" names the discipline of filling the window with the right information rather than crafting one instruction [9]. Anthropic's Model Context Protocol standardises how models reach external tools and data [10] and is complementary to file-first context delivery: CLAW uses such connectors inside stages while the file structure decides what context a stage receives. CLAW's routing-before-loading and on-demand skill loading are direct applications of the lost-in-the-middle finding: the cheapest way to avoid irrelevant context is not to load it.

3 System Architecture

3.1 Design principles

CLAW is built on the same plain-text, one-job-per-unit, configure-the-factory principles that ICM articulates [1], plus four it adds for autonomous multi-domain operation: route before you load; match the tool to the job and escalate only on a named trigger; never spend or act irreversibly without a budget check and, for external actions, a human confirmation; and write anything that must survive the session to a file, because chat is ephemeral. These principles are themselves stored as always-loaded markdown files (`SOUL.md`, `AGENTS.md`) that bind the orchestrator's behaviour.

Table 1: The layered hierarchy in ICM and in CLAW.

Layer	ICM artefact	CLAW artefact	Resident at session start
0 Identity	`CLAUDE.md`	`IDENTITY.md`, `SOUL.md`, `CLAUDE.md`	Yes
1 Routing	`CONTEXT.md`	`AGENTS.md` + router protocol	Yes
2 Stage contract	stage `CONTEXT.md`	`SKILL.md` (frontmatter + on-demand body)	Frontmatter index only
3 Reference	`references/`, `_config/`	brand guides, `MEMORY.md`, FTS5 archive	Index only; full store on demand
4 Working	`output/`	`outbox/`, `today.md`, in-flight plans	The active work-queue only

3.2 The layered context hierarchy

CLAW keeps ICM's five-layer hierarchy and maps it onto an operating system rather than a single pipeline (Table 1). Layer 0 is identity (`IDENTITY.md`, `SOUL.md`). Layer 1 is routing (`AGENTS.md` plus the router). Layer 2 is the stage contract, realised as a `SKILL.md` file with a typed frontmatter and an on-demand body. Layer 3 is stable reference material (brand guides, conventions, durable memory). Layer 4 is per-run working artifacts (the `outbox/`, the daily log, in-flight task files). The crucial property CLAW inherits from ICM is that Layer 2 skill bodies and Layer 3 reference material are not resident; they load only when routing selects them. Section 4 quantifies the effect.

3.3 Intent routing

Every non-command user message is first classified by a lightweight router agent that returns one of four routes: a direct answer, a single named worker, a full orchestration, or a refusal. Routing happens before any skill body or reference file is loaded. This is the mechanism that lets CLAW carry 94 skills and 15 agents without paying their context cost at rest: the router decides which, if any, to load. It is the operating-system analogue of ICM's human deciding which workspace and stage to run, automated so the system can also act on its own schedule.

3.4 Typed agent composition

Where ICM uses one agent reading different files, CLAW composes a small set of typed agents, each defined by a markdown file that pins its model, its allowed tools, and its remit. The roster includes a router, an orchestrator (the planner and synthesiser), parallelisable research workers, a writer, repo-aware coders, calendar and email agents, a finance agent with read-only credentials, domain specialists for the jewellery brand, and a memory curator. The orchestrator never does heavy work itself; it plans, delegates, and synthesises. Workers return summaries, never raw tool output. This is a shallow tree with file handoffs, not a peer message bus, which is why CLAW does not claim AutoGen's tight-loop collaboration. The composition is governed by an escalation ladder (Section 3.5) and a budget ladder (Section 3.6).

3.5 The escalation ladder

A standing rule tells the orchestrator to use the cheapest mechanism that does the job and to escalate only on a named trigger: a direct tool call by default; a script when a result would exceed roughly two thousand tokens or when more than five similar calls are needed; a skill

when a procedure recurs or carries reusable domain context; a subagent only when a genuinely separate context window or synthesis judgement is required. The ladder includes a downgrade duty: work specified at too high a rung is stepped down before acting. This paper's own evaluation tooling is an instance: a single extraction script replaced what would otherwise have been hundreds of file reads.

3.6 Budget governance

CLAW runs against a subscription model with finite five-hour and weekly windows. A budget file records the current mode on a four-level ladder. Green is normal operation. Yellow forbids the most expensive model and halves research parallelism. Orange restricts workers to the cheapest model and one at a time. Red refuses all non-critical work. The ladder is read at session start and binds every subsequent decision, including the escalation ladder, which may not escalate under Orange or Red. The driving signal is the worst of two limits, the rolling weekly usage and the current five-hour window, with targets calibrated to the platform's own usage meter; the system tracks the five-hour window's used capacity separately from its time-to-reset, so a near reset (which restores capacity) is never mistaken for exhaustion. We are not aware of a comparable explicit cost-governance layer in the file-first or multi-agent literature; it is a requirement that emerges only when an agent system runs continuously against a metered budget rather than per-invocation.

3.7 Temporal memory

Durable memory is a layered store, all of it human-readable markdown. An append-only daily log captures the day. A recent-detail file holds durable facts. An always-loaded index lists live entries by title so that full detail is fetched only on demand. Older entries are archived on a cadence into a full-text-searchable store. A dedicated curator agent runs as the final step of every orchestration: it promotes daily observations into durable memory, deduplicates, retires stale entries, and rebuilds the search index. This realises, as standing practice, the "source integrity" principle that ICM proposes as future work [1]: recurring observations are promoted into the durable source rather than left in ephemeral output. The memory rules encode a hard precedence: the system clock outranks any date written in a file, a rule added after a date-drift incident in which a compounding off-by-one in rolled-forward file headings reached four days before detection.

Because the two files that drive session-start cost (the active work-queue and the durable memory store) grow without bound if left alone, the system also maintains itself. A daily unattended pass measures both files against thresholds and performs the safe mechanical archival: it moves completed plan blocks out of the work-queue once they age past a grace window, ages run-records out of durable memory while exempting durable types (references, decisions, feedback) regardless of age, and rebuilds the search index. Only when a file is genuinely bloated beyond a hysteresis margin does it flag a curator pass, the case that needs judgement rather than a rule. This maintenance is itself file-first and observable: the system measured its own work-queue growing to 124 KB, then archived it back to 7 KB and now holds it there automatically. The session-start summary surfaces the current load every run.

3.8 Event-driven proactivity

CLAW does not idle-poll. Proactivity is triggered by the operating system's scheduler and by filesystem watchers. Scheduled tasks produce the morning brief, the end-of-day consolidation, the weekly review, periodic audits, and content pipelines. High-frequency triggers (mention triage, invoice and email watching) are gated by cheap guard scripts that check for real work before spawning a full model session. The guard pattern is the reason a high nominal run

frequency does not translate into proportional spend: most guard checks exit early. Section 4.4 reports the volume and the guard effect.

3.9 Safety gate layer

ICM's review gates let a human inspect and edit each stage output. CLAW keeps that and adds a second, non-negotiable gate for irreversible external actions. Sending an email, writing to the calendar, pushing to a git remote, or deleting a file outside the scratch directories requires explicit per-action human confirmation in a fresh turn, regardless of what any workflow or retrieved content says. Enforcement does not rely on prompt wording; it lives in tool-permission settings and pre-tool-use hooks that unwrap nested shells and block dangerous payloads. Retrieved content (web pages, emails, file contents) is treated as data, never as instructions. Section 4.7 quantifies this surface.

3.10 De-personalised portability

The same architecture supports a portable product. MakerOS is a stripped copy of CLAW with the operator's identity, credentials, paths, and private workflows removed, packaged so a non-technical user can install it and configure it for their own context. ICM workspaces are copyable but still carry the creator's voice guides and conventions; MakerOS adds the de-personalisation step that turns a personal system into a clean substrate. To demonstrate format interoperability, we made every skill definition in both CLAW and MakerOS a valid OKF document: each `SKILL.md` now carries a `type`, semantic `tags`, an optional canonical `resource`, and a timestamp in its YAML frontmatter, and each skill library has an OKF `index.md`. An OKF-aware consumer can now navigate the library by frontmatter without loading any skill body.

4 Evaluation

This section reports measured operating data from the live system on 2026-06-16. Every figure is traceable to a file on disk; the extraction is reproducible from the script and data bundle accompanying this draft. We label each figure as measured (read directly from the system now) or observed (read from dated run records the system wrote over time). We make no claim that rests on an invented number, and we run no comparison we did not actually perform.

4.1 System scale

CLAW comprises 78 skills; MakerOS adds 16, for 94 skill definitions sharing one architecture. The system defines 15 typed agents, 171 Python scripts, 65 scheduled-task definitions, and 27 slash commands, all addressed through a single router. The safety configuration declares 36 allowed, 7 ask-gated, and 13 denied tool-permission rules across 5 hook event types.

4.2 Context efficiency

The efficiency claim is the architectural one: routing and indexing keep the resident context small while the bulk of system knowledge stays on disk until needed. We measured the entire always-resident framing surface (identity, routing, rules, the memory index, the active work-queue, and the daily logs) at 42,431 characters, approximately 10,600 tokens. The durable-memory index is 8,423 characters against a full memory store of 43,935 characters: the always-loaded index is 19 per cent of the full store, and the remaining 81 per cent loads only on demand or through full-text search. The 78 skill bodies and 15 agent definitions, which hold the majority of system knowledge, contribute zero resident tokens until routing selects one.

Table 2: Resident context against a monolithic baseline for the CLAW deployment (token estimates at 4 characters per token).

Component	Resident in CLAW	In a monolithic prompt
Identity, routing, rules	∼5,300 tok	∼5,300 tok
Active work-queue	∼1,700 tok	∼1,700 tok
Durable memory	∼2,100 tok (index)	∼11,000 tok (full store)
78 skill bodies	0 (on demand)	∼105,200 tok
15 agent definitions	0 (on demand)	∼10,100 tok
Daily logs	∼1,400 tok	∼1,400 tok
Total resident surface	∼10,600 tok	∼134,700 tok

For a monolithic baseline we summed what a naive single-prompt system would carry for the CLAW deployment: the framing surface, with the memory index replaced by the full store, plus all 78 skill bodies and all 15 agent definitions. That baseline is approximately 134,700 tokens. CLAW's design holds the resident surface to roughly 7.9 per cent of it, a thirteen-fold reduction (Table 2). The work-queue is included on both sides here, not excluded, because the automated memory hygiene described in Section 3.7 keeps it small: the same instrumentation that measured it growing to 124 KB now archives completed plans daily and holds the live file near 7 KB. The earlier bloat is therefore a maintenance state the system now corrects on its own, not a standing caveat on these figures.

4.3 Operational reliability

The system writes a checkpoint per skill recording consecutive failures, last exit code, and last error. Across 50 skill checkpoints, 49 stood at zero consecutive failures, a 98 per cent clean rate. The single non-clean checkpoint (the self-review skill) recorded one consecutive failure whose stored error was a transient network socket closure, not a logic fault. This is a reliability signal from the system's own bookkeeping, not from external instrumentation, and should be read as such.

4.4 Autonomous run volume and the guard pattern

The scheduled-run log holds 5,183 records spanning 41 days, an average of 126 runs per day. The distribution is dominated by the two guard-gated high-frequency triggers: mention triage (2,145 records) and invoice watching (1,882 records), followed by email watching (480). The remaining roughly 676 records are the substantive scheduled skills: email triage (78), self-review (60), end-of-day consolidation (39), image optimisation (39), and a long tail of audits, content pipelines, and reviews. The shape of this distribution is the point. The high-frequency entries are mostly guard checks that exit without spawning a full model session; the substantive work is a smaller, scheduled set. A high nominal run count does not imply proportional model spend, because the guard pattern (Section 3.8) gates the expensive path. Separately, 484 email-watch poll logs record the filesystem-and-classification watcher operating independently of the model sessions.

4.5 Human-in-the-loop resolution

CLAW surfaces proactive work through a daily action sheet delivered to the operator. Across 20 logged day-sheets carrying 60 items, the operator marked 23 done and 7 skipped, a 50 per cent resolution rate, leaving 30 open. The modest, honest reading is that the system proposes and the human disposes: roughly half of proactively surfaced items are acted on, which is consistent with ICM's finding that human intervention concentrates where judgement is required and

tapers elsewhere [1]. The resolution rate is a usage signal, not a quality score; an unresolved item may be deferred rather than rejected.

4.6 Task-type breadth

Categorising the durable-memory run records by domain shows the breadth of autonomous work the single architecture carries: e-commerce and SEO, image optimisation, finance, design and creative, video and content, multi-channel publishing, task hygiene, infrastructure, personal logistics, and planning, across roughly fourteen distinct categories. No single domain dominates. This breadth, carried by one router over 94 skills, is the operating-system claim made concrete: the same file-first substrate runs a business's content pipeline and the operator's meal plan without separate frameworks.

4.7 Safety surface

The confirmation and isolation surface is explicit and countable: 36 allowed tool rules, 7 ask-gated rules covering exactly the irreversible external actions (email send, calendar write, certain label and file operations), and 13 denied rules. Five hook event types (session start, pre-tool-use, post-tool-use, subagent stop, and stop) carry enforcement that does not depend on prompt wording. The send path for all three email accounts is funnelled through a single ask-gated tool; no agent, including the autonomous watchers, can send mail without a fresh-turn human approval.

5 Discussion

CLAW's position relative to ICM and OKF is that of an existence proof for the next question. ICM shows that one agent and a folder hierarchy suffice for a sequential, human-reviewed pipeline, and scopes out concurrency, autonomy, and branching. OKF shows that a minimal markdown format suffices to share knowledge between agents. CLAW takes both as foundations and reports what happens when the file-first approach is pushed into a long-running, multi-domain, partly autonomous, budget-bound, safety-critical setting. The answer is that the foundations hold, but five additional layers become necessary: routing so the system can decide what to load and act on its own initiative; typed composition so different jobs get different tools and budgets; a cost ladder so a metered budget is respected; temporal memory so knowledge survives and improves across sessions; and a safety gate so autonomy never crosses into irreversible action without a human.

The OKF-compatibility result is worth drawing out. Making 94 skill definitions valid OKF documents and adding library indices cost a single additive metadata pass and broke nothing, because the skills were already markdown files with frontmatter. This is direct evidence for OKF's central claim that the format is the interoperability surface: a system built before the format could be made to speak it without restructuring. It also means a MakerOS install is now a valid OKF bundle, navigable by any OKF-aware tool, not only by the model that wrote it.

The deeper observation echoes ICM's on observability. Because every layer of CLAW is a plain file, the system is inspectable by default, including in ways its designer did not anticipate. The evaluation in Section 4 was possible because the operating record is files, not opaque internal state, and the work-queue bloat in Section 4.2 was both found and then automatically corrected by the same property. A glass-box system is one you can audit, and one that can audit and maintain itself.

6 Limitations and Threats to Validity

This evidence is the detailed operating record of one system used by one operator, and several limitations bound what can be concluded from it.

It is single-operator and single-author. There is no second user, so the resolution rates, reliability figures, and usage patterns reflect one person's workflows and tolerances, with the attendant enthusiasm and selection bias.

It is single-model-family. Like ICM, all operation has been on one provider's models. The architecture is designed to be model-agnostic, but cross-model behaviour is untested and is a clear next step.

It is observational, not controlled. We ran no A/B comparison between CLAW's scoped loading and a monolithic prompt on the same tasks. The context-efficiency figures are static measurements of what loads, and the monolithic baseline is a construction of what a naive system would carry, not a measured competitor. The reliability and resolution figures come from the system's own bookkeeping, not from independent instrumentation. The lost-in-the-middle support for scoped context [7] is borrowed, not re-measured here.

Some headline counts mix kinds. The 5,183 scheduled records are dominated by lightweight guard checks; we report the breakdown precisely so the figure is not read as 5,183 full model sessions. The 126-runs-per-day average is similarly a count of triggers, most of which are guards.

The work-queue bloat that the system's instrumentation surfaced (Section 4.2) showed that a manual archival cadence is not reliably followed. We treat this as an operational finding, distinct from the architecture, and it has since been addressed by the automated hygiene pass (Section 3.7). We report both the original lapse and the fix because the file-first observability is what made the lapse measurable in the first place.

A controlled user study, cross-model evaluation, and a second operator would each substantially strengthen the empirical foundation. None is present here.

7 Future Work

Three directions follow directly. First, cross-model evaluation: run the same skills and the same router under a different model family and measure where behaviour diverges. Second, semantic debugging and source integrity, which ICM proposes for sequential pipelines [1] and which CLAW already begins through curator-driven promotion of recurring observations into durable memory; the open problem is tracing a flawed output back through routing, skill, and memory to the source that produced it. Third, a controlled comparison of scoped against monolithic loading on identical multi-domain tasks, to replace the constructed monolithic baseline with a measured one. Beyond these, a structured study of MakerOS users would test whether the de-personalised substrate transfers to operators who did not build it.

8 Conclusion

The filesystem is a sufficient substrate for far more than a single sequential pipeline. ICM showed it can replace framework orchestration for one agent and one human-reviewed sequence. OKF showed a minimal markdown format can carry shareable knowledge between agents. CLAW shows that the same file-first foundation, extended with routing, typed composition, a cost ladder, temporal memory, and a safety gate, can run a long-lived, multi-domain, partly autonomous personal operating system on a consumer machine against a subscription model, while staying inspectable, governable, and safe. The architecture is plain files. The control surface is a text editor. The evidence is one system's month of operation, reported in full, limits included.

Acknowledgements

The evaluation tooling and the metric extraction in Section 4 were generated by the CLAW system itself under the author's direction, an instance of the glass-box property the paper describes.

References

[1] J. Van Clief and D. McDermott. Interpretable Context Methodology: Folder Structure as Agent Architecture. arXiv:2603.16021v2 [cs.AI], 2026.

[2] Google Cloud. How the Open Knowledge Format can improve data sharing. Google Cloud Blog, 2026. Repository: GoogleCloudPlatform/knowledge-catalog.

[3] S. Hong, X. Zheng, J. Chen, et al. MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework. ICLR 2024, arXiv:2308.00352.

[4] Q. Wu, G. Bansal, J. Zhang, et al. AutoGen: Enabling Next-Gen LLM Applications via Multi-Agent Conversation. COLM 2024, arXiv:2308.08155.

[5] C. Packer, S. Wooders, K. Lin, et al. MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560, 2023.

[6] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative Agents: Interactive Simulacra of Human Behavior. UIST 2023. doi:10.1145/3586183.3606763.

[7] N. F. Liu, K. Lin, J. Hewitt, et al. Lost in the Middle: How Language Models Use Long Contexts. TACL, vol. 12, pp. 157–173, 2024. arXiv:2307.03172.

[8] J. Wei, X. Wang, D. Schuurmans, et al. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. NeurIPS 2022. arXiv:2201.11903.

[9] A. Karpathy. +1 for context engineering over prompt engineering. X, 25 June 2025.

[10] Anthropic. Introducing the Model Context Protocol. Anthropic Blog, 25 November 2024.

[11] T. Wu, M. Terry, and C. J. Cai. AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts. CHI 2022. doi:10.1145/3491102.3517582.

[12] C. Rudin. Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. Nature Machine Intelligence, vol. 1, pp. 206–215, 2019.

[13] B. Shneiderman. Human-Centered Artificial Intelligence: Reliable, Safe and Trustworthy. International Journal of Human-Computer Interaction, vol. 36, no. 6, pp. 495–504, 2020.

[14] M. D. McIlroy, E. N. Pinson, and B. A. Tague. Unix Time-Sharing System: Foreword. The Bell System Technical Journal, vol. 57, no. 6, pp. 1899–1904, 1978.