

Architecting Domain-Embedded AI Agents: Memory, Tools, Governance, and Multi-Surface Orchestration

Research Note #001

Benjamin Koper

Clippable Research · United States

Version of record: June 4, 2026

DOI: <https://doi.org/10.5281/zenodo.20549039>

<https://clippable.io/research/ai-social-media-agents-system-design>

Abstract

Most production “agents” today are still chat sessions with extra API calls. That is enough for drafting copy; it is not enough when work has owners, deadlines, spend limits, and an audit log. This research note lays out a practical architecture for domain-embedded agents: the model reads and writes through product schemas, keeps memory outside the prompt, routes side effects through tools, and leaves humans on the hook for irreversible steps. The argument is grounded in published work on tool-augmented language models and in operational constraints from social-media programs, where coordination cost often exceeds model capability. The note is descriptive engineering guidance, not a benchmark study.

1. Session chat stops where operations begin

A general chat window has no memory of what went live last Tuesday, which variant won, or who approved a claim. Operators compensate by pasting briefs, linking spreadsheets, and repeating context in every thread. That workaround scales poorly once more than one person touches the same program.

Embedding the agent in product state changes the contract. Campaigns, roles, assets, and integration handles become the interface the model sees. The user still chats, but the system of record is not the transcript. Continuity survives tab closes because state lives in the workspace, not in the last prompt.

2. Memory: keep hot state small

Useful memory is not “store everything.” In practice we see three layers that are easy to confuse. Working context holds the active plan and fresh tool output. Episodic traces store prior runs with enough metadata to answer “what did we try?” Semantic stores hold slower-moving facts: positioning, catalog rows, policy snippets.

Blind retrieval hurts. Stuffing long histories into the context window raises cost and makes confabulation more likely, especially when old instructions conflict with new ones. A workable default is tiered compaction: keep the current task hot, summarize the last few weeks into retrievable chunks, and archive the rest behind explicit search. Retention policy matters as soon as the store holds customer or creator data; memory design is a privacy decision, not an afterthought.

3. Tools, planners, and when to split roles

Tool catalogs grow faster than any single prompt can supervise. Monolithic “do everything” agents tend to misuse tools under load or skip verification when latency spikes. A pattern that has held up in production is boring on purpose: a planner proposes structured steps, narrow specialists call APIs they understand, and a critic checks policy before commits.

Tool design carries most of the reliability. Idempotent calls, explicit error codes, and dry-run modes beat clever prompting. In social workflows you also inherit messy peripherals (media generation, schedulers,

payout rails), each with different failure semantics. The orchestrator should own retries, partial success, and escalation; expecting the base model to infer that from tone alone fails in the boring cases, which are the ones that show up in incident reviews.

4. Governance without theater

Full autonomy sounds efficient until an off-brand post, overspend, or policy miss lands in legal review. Teams that ship safely default to propose-and-approve: the agent prepares a diff, a human approves or edits, and both versions are logged. Role-based tool access, spend caps, and classifiers are unglamorous and necessary.

Reviewer edits are signal, not noise. The hard part is capturing them per workspace without leaking preferences across customers, still an open problem for enterprise deployments, adjacent to preference learning work but with stricter isolation requirements than public RLHF pipelines assume.

5. One agent, several surfaces

Operators do not experience “the web product” and “SMS” as separate products. They expect the same identity, permissions, and thread of work. That only works if session keys, policy checks, and tool backends are shared under surface-specific adapters. Voice and text impose shorter turns and tighter latency budgets than a desktop planning UI; the core state machine can stay shared while the presentation layer changes.

Vertical specialization is coming (commerce, healthcare, public sector), but most teams will reuse the same boring infrastructure: auth, observability, evaluation harnesses. Context window size helps; it does not replace those pieces.

6. Limits of this note

We do not report new benchmarks or ablations here. The contribution is a consolidation of design choices seen while building governed agents for marketing operations, mapped to existing literature on tool use and multi-step control. Follow-on work from this lab will focus on evaluation: tying agent plans to outcome data without turning every report into a vanity metric.

References

1. Schick, T., et al. (2023). Toolformer: Language models can teach themselves to use tools. arXiv:2302.04761. <https://doi.org/10.48550/arXiv.2302.04761>
2. Yao, S., et al. (2022). ReAct: Synergizing reasoning and acting in language models. arXiv:2210.03629. <https://doi.org/10.48550/arXiv.2210.03629>
3. Mialon, G., et al. (2023). Augmented language models: a survey. arXiv:2302.07842. <https://doi.org/10.48550/arXiv.2302.07842>
4. Anthropic. (2024). Building effective agents. <https://www.anthropic.com/engineering/building-effective-agents>
5. NIST. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0). <https://www.nist.gov/itl/ai-risk-management-framework>