Agent-Oriented HATEOAS vs. Static Tool Registries: An Empirical Study of Token Efficiency and Sequencing in Autonomous LLM Agents

Amrit Pandey
mail.amritpandey@gmail.com

Abstract

Large language model (LLM) agents increasingly interact with external systems through *tool registries*—structured catalogs of callable functions exposed at session start. The Model Context Protocol (MCP) has emerged as a popular standard for such registries, but it inherits a static discovery model: every registered tool is typically injected into the agent context on every turn, regardless of workflow state. We introduce *Agent-Oriented HATEOAS* (AOH), a hypermedia-driven alternative in which REST responses carry *affordances*—machine-readable descriptions of only the actions valid in the current state. We implement a reproducible travel-orchestration benchmark (flight, transit, payment) with a shared backend, comparing AOH against MCP under identical agent policies and models. Across 30 trials per condition, AOH reduces median total token consumption by 1.8× relative to a lean seven-tool MCP registry and by 6.5× when the MCP registry is inflated to 35 tools with distractors. Both protocols achieve near-perfect task completion on `gemini-2.5-flash`, and invalid-sequence error rates remain statistically indistinguishable. Our results support AOH as a practical mechanism for *context-efficient* agent integration, while sequencing robustness under registry bloat requires further study with weaker models and larger tool sets.

Keywords: LLM agents, HATEOAS, MCP, tool use, hypermedia, REST, token efficiency

1 Introduction

Autonomous LLM agents depend on external tools to search databases, invoke APIs, and mutate application state. Two design questions dominate production deployments: *what* tools should an agent see, and *when* should it see them? Contemporary agent stacks—including OpenAI function calling, LangChain tool bindings, and Anthropic's Model Context Protocol (MCP) [3]—most often answer the second question with *upfront static discovery*: the full tool catalog is serialized into the model prompt at the beginning of a session and re-sent on subsequent turns.

Static discovery is simple to implement and interoperable across hosts, but it scales poorly with registry size. Each tool contributes a name, natural-language description, and JSON Schema parameters to the prompt budget. In enterprise settings, registries routinely contain dozens or hundreds of overlapping capabilities (payments, bookings, CRM, analytics), many of which are invalid or meaningless in the agent's current workflow phase. The agent must therefore spend context tokens on irrelevant tools and rely on implicit reasoning to infer valid sequencing—a burden that grows with registry cardinality.

Hypermedia architectures offer an older but principled alternative. Fielding's REST dissertation [1] argues that clients should navigate application state through typed links and forms embedded in representations, rather than out-of-band knowledge of a fixed endpoint catalog. Hypermedia As The Engine Of Application State (HATEOAS) constrains the client to *currently valid* transitions, reducing invalid requests by construction. We adapt this idea to LLM agents as **Agent-Oriented HATEOAS (AOH)**: API responses include an `affordances`

array describing only the actions legal in the present state, which the agent runtime converts into dynamic function declarations.

This paper makes three contributions:

1. A formal problem statement for comparing static MCP-style registries against dynamic AOH affordances in multi-step agent workflows.
2. An open benchmark implementing both protocols over an identical Flask/SQLite state machine, with telemetry for tokens, tool calls, sequencing errors, recovery, and latency.
3. An empirical evaluation on `gemini-2.5-flash` showing substantial token savings (1.8–6.5× at median) with unchanged AOH cost as MCP registries grow, alongside an honest null result on invalid-sequence differentiation at the tested scale.

2 Problem Definition

We study autonomous agents that must complete a *goal-directed workflow* $W = (S, s_0, s^\star, \mathcal{A})$ where S is a finite state space, s_0 the initial state, $s^\star$ a terminal success state, and $\mathcal{A}$ a set of actions with preconditions $pre(a) \subseteq S$ and effects $post(a) : S \to S$.

At each turn t, the agent observes an *observation* o_t (task prompt, prior tool results, and optionally resource state) and selects an action $a_t \in \mathcal{A}$. The action is valid iff $s_t \models pre(a_t)$; invalid selections constitute *sequencing errors*. The agent succeeds if it reaches $s^\star$ within a step budget $T_{\max}$.

We isolate **tool exposure policy** as the independent variable:

- **Static registry (MCP-style).** At every turn, the agent receives declarations for a fixed set $\mathcal{T}_{\text{static}} \supseteq \mathcal{A}$, independent of s_t.
- **Dynamic affordances (AOH).** At turn t, the agent receives declarations for $\mathcal{A}_t = \{a \in \mathcal{A} \mid s_t \models pre(a)\}$ only, as embedded hypermedia in the latest API response.

All other factors—backend logic, stochastic failures, base model, system instruction, and task wording—are held constant.

2.1 Metrics

Given a trial τ with prompt tokens P_τ, output tokens O_τ, tool calls N_τ, and error counts, we report:

- **Success rate:** fraction of trials reaching $s^\star$.
- **Total tokens:** $P_\tau + O_\tau$ (primary efficiency metric).
- **Invalid-sequence events:** tool invocations rejected due to precondition violations or out-of-state calls.
- **Recoverable errors:** transient failures (e.g., resource contention) where retry is expected.
- **Distractor calls** (MCP hard condition only): invocations of non-canonical decoy tools.
- **Elapsed wall time** per trial.

2.2 Hypotheses

H1 (Token efficiency).
AOH consumes fewer prompt tokens than MCP because it omits irrelevant tool schemas from context.

H2 (Sequencing robustness).
AOH incurs fewer invalid-sequence events because affordances are filtered by state.

H3 (Registry scaling).
As $|\mathcal{T}_{\text{static}}|$ increases, MCP token cost grows while AOH cost remains stable; the gap widens.

3 Related Work

REST, Richardson maturity, and HATEOAS. Richardson's REST maturity model [2] positions HATEOAS at the highest level: clients discover transitions from representations rather than hard-coded URLs. Fielding [1] emphasizes that hypermedia controls application flow and enables evolvable servers. While HATEOAS saw limited adoption in human-facing SPAs, its constraint—*only advertise valid next steps*—maps naturally onto stateful agent workflows.

LLM tool use and function calling. Instruction-tuned models can select structured tool calls from JSON schemas [4, 5]. Agent frameworks (AutoGPT, LangChain, CrewAI) wrap tools as static Python functions or OpenAPI operations. Recent work on tool retrieval [6] addresses large catalogs by *selecting* a subset per query, but still treats selection as a separate retrieval stage rather than a server-driven hypermedia contract.

Model Context Protocol. MCP [3] standardizes how hosts expose tools, resources, and prompts to LLM clients over stdio or HTTP. Tool definitions are registered at connection time and listed via `tools/list`. Our MCP baseline follows this pattern: the runner fetches the full tool list once and supplies all declarations on every model call. MCP improves interoperability but does not prescribe state-conditioned tool surfacing.

Agent-oriented APIs and hypermedia for machines. Prior work on conversational REST [8] and Linked Data APIs explores machine-navigable interfaces. AOH differs by targeting *LLM function-calling runtimes*: affordances are JSON-Schema-shaped declarations consumable directly by Gemini/OpenAI-style APIs, updated after each state transition.

Evaluation of agent reliability. Benchmarks such as WebArena [9] and ToolBench [7] measure task success on diverse tools. We deliberately use a minimal domain with a known valid sequence to *isolate* exposure policy rather than measure open-world generalization.

4 Background

4.1 REST APIs

Representational State Transfer (REST) [1] treats servers as state machines addressed through uniform HTTP verbs over resource identifiers. A *representation* (typically JSON) captures resource state; *hypermedia controls* (links, forms) describe how to transition to related resources. Clients that honor hypermedia need not embed a complete API map—they follow controls present in the current representation.

Table 1: Static MCP registries vs. Agent-Oriented HATEOAS.

Aspect	**MCP (static registry)**	**AOH (dynamic affordances)**
Discovery	Once at session start; full list	Per response; state-filtered
Context per turn	$O(\lvert\mathcal{T}_{\text{static}}\rvert)$ schemas	$O(\lvert\mathcal{A}_t\rvert)$ schemas
Sequencing hint	Implicit in descriptions	Explicit in affordance set
Transport	stdio/HTTP MCP	REST/HTTP JSON
Invalid action	Runtime error from server	Often absent from declarations

4.2 HATEOAS

HATEOAS requires that application state changes be driven exclusively through hypermedia. Practically, a response might include:

```
{
  "session": {"current_state": "flight_booked", ...},
  "affordances": [
    {"name": "book_ride", "method": "POST",
     "href": "/sessions/.../transit/ride", ...}
  ]
}
```

Only actions whose preconditions hold in `flight_booked` appear. Invalid transitions are rejected server-side, but the primary agent-facing benefit is *context reduction*: the model never sees payment tools before transit is booked.

4.3 Model Context Protocol (MCP)

MCP defines a client–host–server architecture for LLM applications. Servers register *tools* with name, description, and input schema. The host aggregates tools and exposes them to the model for function calling. Discovery is typically *session-static*: `tools/list` returns the complete registry; hosts may filter, but the default pattern injects all tools into each inference request.

Table 1 summarizes the conceptual contrast.

5 Agent-Oriented HATEOAS

5.1 Design

AOH applies HATEOAS to LLM agents as follows:

1. The server maintains workflow state s_t for each session.
2. Every AOH response includes `affordances`: an array of action descriptors compatible with the agent runtime's function-calling format (name, description, HTTP method, href, JSON Schema parameters).
3. The agent runner converts `affordances` into ephemeral tool declarations for the *next* model call only.
4. After a successful invocation, the server returns updated state and a fresh affordance list; failed invocations return errors but may retain the same affordances for retry.

Payment affordances, for example, are withheld until `current_state = transit_booked`, mirroring business preconditions without relying on the model to infer them from static tool docs.

AOH (per step): *init* → search/book flight tools only; *flight_booked* → transit tools; *transit_booked* → payment tool.

MCP (every step): all 7 (baseline) or 35 (hard) tool schemas in context.

Figure 1: Tool exposure model: AOH affordances shrink with state; MCP sends the full registry each turn.

5.2 Comparison Against MCP in This Benchmark

Both protocols invoke the same Python business functions (`search_flights`, `book_flight`, `check_transit_options`, `book_ride`, `process_payment`) backed by SQLite. The MCP server wraps them as FastMCP tools; the AOH server exposes them as REST routes with dynamic affordances.

Fairness controls. The evaluation runner (`runner.py`) applies an identical `SYSTEM_INSTRUCTION` to both modes: act autonomously, select cheapest valid options, retry transient failures, and continue until payment succeeds. Text-only model turns trigger a continuation nudge rather than ending the trial. The only deliberate difference is tool exposure policy.

MCP conditions. We evaluate two registry sizes:

- **Baseline (7 tools):** canonical workflow tools only.
- **Hard (35 tools):** 7 canonical plus 28 distractors—12 same-domain decoys (`request_ride`, `checkout`, etc.) with overlapping descriptions and 16 cross-domain noise tools (`search_hotels`, `get_weather_forecast`, etc.), simulating enterprise registry sprawl.

AOH is unchanged across conditions; it never advertises invalid or decoy actions.

5.3 Workflow State Machine

The benchmark domain is a DEL→BLR travel workflow with states *init* → *flight_booked* → *transit_booked* → *paid*. Ride booking fails stochastically with probability 0.3 ("No drivers available") to test recovery. Figure 1 illustrates exposure differences at each state.

6 Experimental Evaluation

6.1 Setup

- **Model:** Google `gemini-2.5-flash` via the Gemini function-calling API.
- **Trials:** 30 paired runs per mode per condition (AOH + MCP), fresh UUID session each trial.
- **Task prompt:** Book DEL–BLR flight, airport-to-hotel ride, pay $420 total; session ID embedded.
- **Step budget:** 12 tool-calling turns maximum.
- **Infrastructure:** Flask AOH server on port 5050; MCP server over stdio; telemetry logged to JSONL.

Figures are generated reproducibly:

```
python scripts/analyze_telemetry.py
python scripts/plot_results.py
```

which produce `figures/fig1_token_cost.pdf` through `fig4_trajectories.pdf`.

Table 2: Experimental results (30 trials per cell, `gemini-2.5-flash`).

Condition	Mode	Med. total tokens	Med. prompt	Success	Inv. seq.
Baseline (7 tools)	AOH	4,539	4,446	100%	0.20
Baseline (7 tools)	MCP	8,191	7,883	100%	0.17
Hard (35 tools)	AOH	4,539	4,446	100%	0.17
Hard (35 tools)	MCP	29,353	29,053	96.7%	0.07

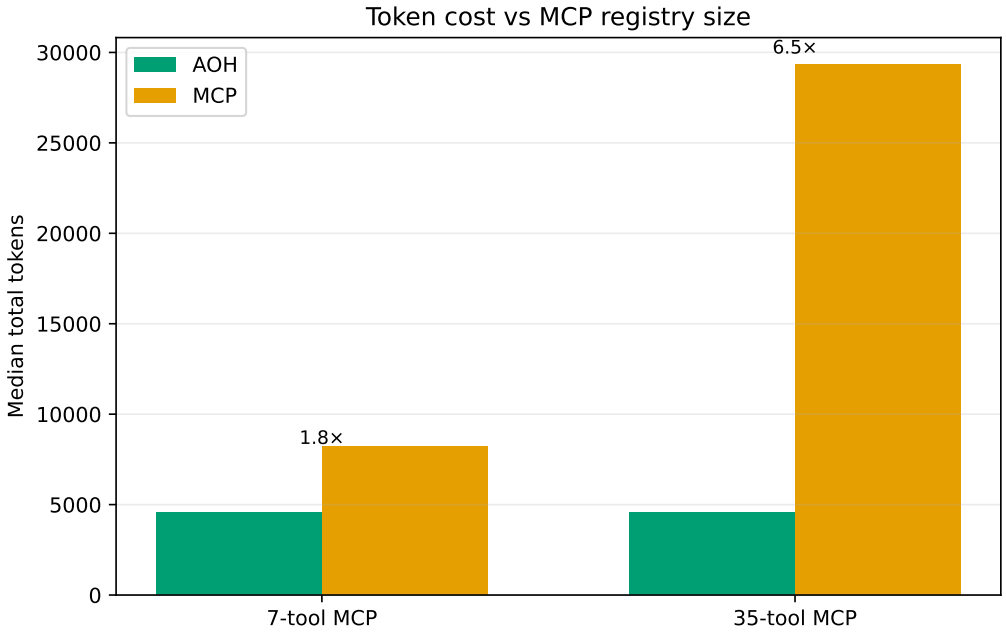

Figure 2: Median total token consumption by protocol and MCP registry size.

6.2 Results

Table 2 reports medians and success rates from the benchmark telemetry (30 clean trials per cell; baseline JSONL excludes two pre-fix pilot records).

Token efficiency (H1, H3). Figure 2 shows median total tokens across conditions. AOH median cost is *identical* (4,539 tokens) in baseline and hard runs, as affordance count stays bounded by workflow phase (2–3 tools per state). MCP median cost rises from 8,191 to 29,353 when distractors are added—a 6.5× increase relative to AOH in the hard condition and 1.8× in baseline. Prompt tokens account for nearly all of the gap; median output tokens remain similar (93 vs. 301–308 for MCP).

Reliability and sequencing (H2). Figure 4 aggregates success, invalid-sequence, recoverable, and distractor metrics. Both protocols achieve ≥96.7% success. Mean invalid-sequence events per trial are near zero (≤0.20) and do *not* favor AOH at this registry scale: with only seven clearly named canonical tools, MCP agents sequence correctly. In the hard condition, MCP distractor calls average 0.03 per trial (one `request_ride` invocation across 30 runs); one MCP trial failed by exhausting the step budget at `transit_booked`.

Per-trial variance. Figure 5 plots total tokens per trial. AOH forms a tight low cluster; MCP hard exhibits higher variance and an outlier corresponding to the failed trial (∼66k prompt tokens).

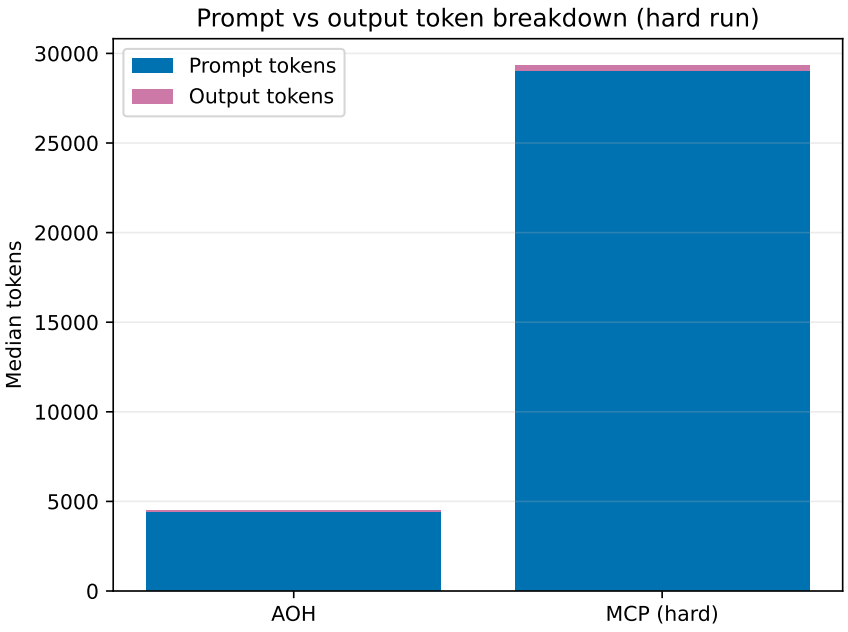

Figure 3: Median prompt vs. output token breakdown (hard condition). MCP prompt bloat dominates.

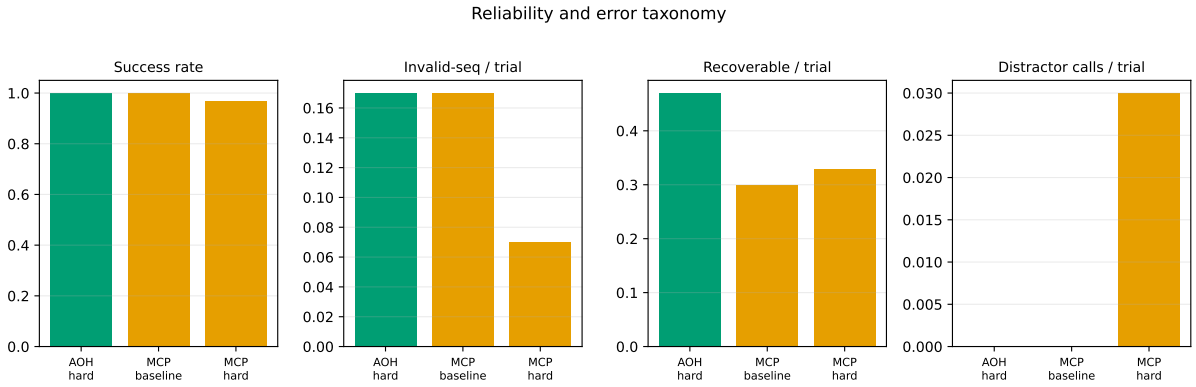

Figure 4: Reliability and error taxonomy across conditions.

6.3 Discussion

Supported claims. H1 and H3 are supported: dynamic affordances materially reduce prompt tokens, and the savings amplify as static registries grow. This has direct cost implications for hosted agents billed per token and for context-window-limited models.

Unsupported claims. H2 is *not* supported at the tested scale with `gemini-2.5-flash`. Modern models with strong tool-selection capability can navigate small canonical MCP registries without measurable sequencing harm; inflating the registry primarily taxes context, not validity. Distractors rarely tempt the model, suggesting that decoy-heavy registries may need weaker models, ambiguous naming, or orders-of-magnitude more tools to stress-test H2.

Threats to validity. Single domain and single model family; stochastic ride failures introduce variance; MCP distractors are synthetic; wall-clock times include network latency. Future work should sweep models (e.g., flash-lite tiers), registry sizes, and domains with longer horizons.

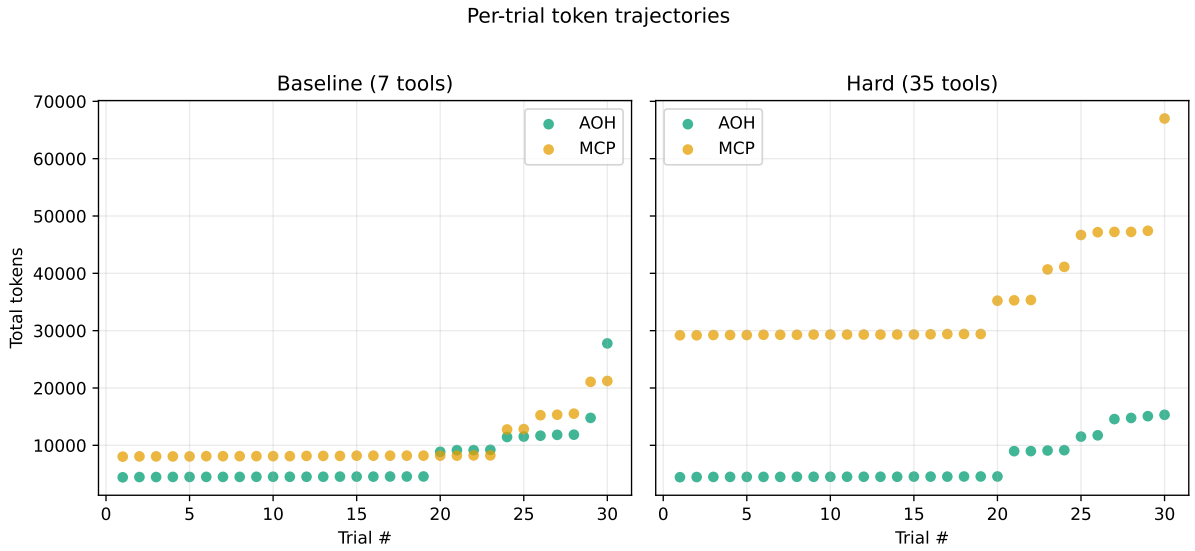

Figure 5: Per-trial total token usage (baseline vs. hard registry).

7 Conclusion

We presented Agent-Oriented HATEOAS, a hypermedia pattern that exposes state-valid affordances to LLM agents instead of static MCP-style tool catalogs. In a controlled travel-orchestration benchmark with matched backend logic and agent policy, AOH reduces median token usage by 1.8× (lean registry) to 6.5× (35-tool registry) while maintaining 100% success, whereas MCP token cost scales with registry size.

AOH does not, in our experiments, reduce invalid tool sequencing relative to MCP when the model is `gemini-2.5-flash` and canonical tools are well named. We therefore position AOH primarily as a **context-efficiency and evolvability** mechanism—servers publish what is legal *now*—rather than as a universal hallucination suppressor.

Practitioners integrating MCP can adopt AOH principles incrementally: return state-specific tool subsets from `tools/list` filters, embed affordances in resource responses, or hybridize MCP transport with hypermedia discovery. We release our benchmark, telemetry pipeline, and Matplotlib figure scripts to support reproducible comparisons.

Future work. Evaluate weaker models where distractor selection may rise; measure very large registries (10^2–10^3 tools); study hybrid MCP+AOH servers; extend to multi-agent and human-in-the-loop settings; and analyze dollar cost and carbon footprint of prompt bloat at production traffic scales.

Acknowledgments

Omitted for anonymous review.

References

[1] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[2] L. Richardson and S. Ruby. *RESTful Web Services*. O'Reilly Media, 2008.

[3] Anthropic. Model Context Protocol specification, 2024. https://modelcontextprotocol.io/.

[4] T. Schick et al. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

[5] OpenAI. Function calling and other API updates, 2023. https://openai.com/index/function-calling-and-other-api-updates/.

[6] Y. Qin et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*, 2023.

[7] Y. Qin et al. ToolBench: A benchmark for tool learning with LLMs. *OpenReview*, 2023.

[8] J. Domingue, M. Fernandez, and A. M. Lopez. APIs for conversational interfaces. In *WWW Companion*, 2018.

[9] S. Zhou et al. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.