toolc: An Optimizing Compiler for Agent Tool Surfaces

August 24, 2026

Abstract

Language-model agents increasingly act through tools exposed via the Model Context Protocol (MCP). In practice, agents connect to several servers at once, and the naive federation of their catalogs scales poorly: every tool definition occupies context, near-duplicate tools degrade selection accuracy, and multi-step workflows must be improvised call-by-call. We present TOOLC, an optimizing compiler that treats the federated tool surface as a compilation target. TOOLC introspects downstream servers (or synthesizes catalogs directly from OpenAPI specifications or human-readable API documentation) into a capability-graph intermediate representation, then applies a sequence of independently toggleable passes: dead-tool elimination, LLM-assisted description rewriting with intra- and inter-server disambiguation, consolidation of near-duplicate tool families into deterministic facade tools, macro inlining, and an optional retrieval-based selection surface. The compiled artifact is served as a single drop-in MCP endpoint with full call logging, serve-time result compaction, and error-driven re-optimization. In an evaluation over 137 tools from six live servers (60 tasks $\times$ 3 trials $\times$ two agent models; 1,800 judged trials), the compiled surface with consolidation matched the raw federation's success for a frontier agent (96.1% vs. 96.7%, overlapping intervals) at 5$\times$ lower cost per completed task, and *improved* cross-source arbitration (90% $\rightarrow$ 100% on overlap tasks); for a small agent the same compression cost nine points, showing the benefit is capability-gated. Every optimization is a measured, reversible configuration choice rather than a fixed architecture.

1 Introduction

Tool use has become the dominant mechanism by which language-model agents act on the world, and the Model Context Protocol has emerged as a standard transport for exposing tools to agents. The unit of deployment, however, is the *server*, not the *task*: a working agent setup typically federates several servers, each contributing tens of tool definitions written by different authors with different conventions that are not designed to work in parallel.

This federation is an unoptimized program. Its costs are threefold:

1. **Context.** Every served tool is prompt input on every turn. In our reference federation, 131 definitions consume roughly 24,000 tokens before the agent reads the request.
2. **Selection accuracy.** Catalogs contain near-duplicates (`search_x`, `find_x`, `get_x`), machine-generated descriptions, and cross-server overlap (two news sources). The agent's tool choice degrades with catalog entropy, and errors cost round trips.
3. **Orchestration.** Common multi-step workflows (find an entity, fetch its records, extract a field) are re-derived by the agent on every occurrence, at inference prices.

Our thesis is that these are compiler problems. The tool surface admits an intermediate representation, semantics-preserving transformations over that representation, and a runtime that executes the compiled program while observing it for further optimization. TOOLC realizes this thesis end to end: it is available as an Apache-2.0 engine and as a hosted service that adds managed endpoints, observability, and usage-based billing.

This paper makes four contributions:

1. A capability-graph IR for federated tool surfaces that preserves original definitions as immutable facts and expresses every transformation as a reviewable overlay (§3).
2. A pass architecture comprising dead-tool elimination, LLM-assisted definition rewriting, deterministic consolidation of tool families into facades (within and across servers), macro inlining, and a retrieval-based selection surface; each pass is independently toggleable and its effect independently measurable (§4).
3. A serving runtime with logged dispatch, serve-time result compaction, and profile-guided re-optimization that converts recurring runtime errors into proposed compiler fixes (§5).
4. A judged, ablated evaluation methodology and pilot results showing that compilation dominates the naive federation on both cost and accuracy axes (§6).

2 Related Work

Tool learning and selection. A substantial literature addresses tool-augmented language models [1, 2] and tool selection at scale, including retrieval over large tool inventories [3, 4] and benchmarks of tool-use competence [5]. Tool documentation itself drives zero-shot usage quality [6], which our rewrite pass operationalizes. Our selection pass adopts retrieval (BM25 [7] over compiled descriptions) as one point on a measured cost/accuracy frontier rather than as the architecture.

API-to-agent interfaces. The Model Context Protocol [8] standardizes the transport this work compiles for. Automatic conversion of OpenAPI specifications into agent tools is widely practiced and widely observed to produce poor surfaces; our composition frontend embraces the mechanical conversion precisely because the compiler owns making the result good, and our evaluation includes a deliberately naive machine-generated server as a worst-case input.

Prompt and context optimization. Description rewriting relates to automatic prompt optimization [9]; result compaction relates to context summarization. TOOLC differs in binding these to a persistent, versioned artifact with provenance (original text is never destroyed) and in validating LLM proposals with deterministic checks before they take effect.

Compiler framing. We borrow the classical structure [10] — frontend, IR, passes, emitter, profile-guided optimization [11] — and find it maps cleanly: tool catalogs are modules, near-duplicate families are common subexpressions, macros are inlined procedures, the selection surface is lazy loading, and runtime error clustering is PGO.

3 The Capability-Graph IR

A compiled surface is a graph $G = (S, T, E)$: sources S (downstream servers with catalog hashes), tools T, and data edges E (produced-by / consumed-by relations contributed by macros).

Each tool node carries:

- identity, the *original* description and JSON input schema, immutable once introspected;

- *overlays*: the rewritten description, visibility, and surface placement (`surfaced`);
- a kind: `passthrough`, `facade`, `macro`, or `meta`;
- for facades, a deterministic routing table from action names to member tool ids.

Two properties matter. **Provenance**: originals are never mutated, so any transformation is auditable and reversible, and user-facing consoles can display exact before/after diffs. **Content addressing**: the graph serializes deterministically and is versioned by content hash, so a served artifact is immutable and cache keys (e.g., for rewrites) are stable.

Frontends produce this IR from three inputs: live MCP introspection, OpenAPI specifications (each operation becomes a passthrough tool with constraints — enums, bounds, defaults — carried into the schema), and human-readable documentation, from which a draft specification is synthesized by an LLM under never-invent rules and then *probe-validated*: parameterless GET endpoints are called against the live API, and endpoints that return 404 are flagged as probable hallucinations before anything compiles for ensure integrity.

4 Optimization Passes

Passes run in a fixed order, each mapping graph to graph; every pass emits a structured diff for review. All passes are user-toggleable configuration.

Dead-tool elimination hides tools matched by user globs or by built-in noise patterns (such as health checks, debug endpoints).

Rewrite regenerates descriptions with an LLM, batched per server with full sibling context plus a digest of the rest of the pool, under instructions to disambiguate against both siblings and overlapping tools on other servers, to surface parameter constraints, and never to invent capabilities. Proposals are cached by (tool, original-description hash, prompt version) and can be gated on human review.

Consolidate merges families of near-duplicate tools into *facade* tools with an `action` discriminator. The LLM only proposes group membership and writes the group description; the compiler validates every proposal (members exist, no reuse, size bounds, no id collisions) and synthesizes the union schema and routing table deterministically, so a poor proposal can be rejected but never mis-routed. Dispatch validates arguments against the *member's* original schema. An opt-in second phase merges overlapping capabilities *across* servers (two news providers) into capability facades whose actions name their source, making vendor choice an explicit, documented decision.

Macro inlining adds hand-authored multi-step tools (typed steps over existing tools) as first-class nodes, contributing data edges to the graph.

Selection replaces the served catalog with two meta-tools — `search_tools` (BM25 retrieval over compiled definitions) and `call_tool` (validated dispatch by id) — shrinking served context to a few hundred tokens regardless of catalog size, at the price of a search round trip.

5 Serving Runtime

The compiled artifact is served as a standard MCP endpoint. Every dispatch is logged (arguments, result, latency, parent call for facade/macro/meta fan-out), giving operators full-payload observability with a live tail for easy debugging.

Result compaction. Results exceeding a token threshold are summarized at serve time by a small model under instructions that preserve identifiers and sourcing metadata (record ids, permalinks, timestamps), with a deterministic structural fallback (bounded arrays, capped strings) when the model is unavailable, and an asynchronous mode above a second threshold that serves a structural cut immediately while caching the model-compacted version for repeat calls. Error results and selection meta-results are never compacted.

Profile-guided re-optimization. Recurring runtime errors are clustered by (tool, normalized error signature); each new cluster is analyzed by an LLM against the tool definition *as agents currently see it*, and the proposal is constrained to a typed action: amend the description (a persistent overlay) or block the tool. Operators apply fixes with one action; the applied overlay survives recompilation. The loop was validated by reproducing, automatically, two fixes we had first made by hand (a missing numeric constraint; a chronically failing tool).

6 Evaluation

Setup. 137 tools across six live servers: a filesystem reference server, the public Hugging Face and GitHub MCP servers, a deliberately naive machine-generated National Weather Service server (one tool per GET operation, verbatim descriptions), and two REST APIs (Open-Meteo, Hacker News search) composed through our OpenAPI frontend — chosen to create genuine cross-server overlap in weather and article search. 60 tasks across six categories (single-tool, ambiguous-selection, cross-server chains, needle-in-haystack, no-tool-exists, and *overlap-selection*: tasks answerable by multiple sources, including sources that *cannot* answer, e.g. the US-only NWS for Paris weather), 3 trials each, two agent models (`claude-sonnet-4-6`, `claude-haiku-4-5`; 1,800 trials total), answers judged by a stronger model against task-specific rubrics (an LLM-as-judge design [12]).

	`sonnet-4-6`		`haiku-4-5`	
Condition	Success	$/task	Success	$/task
Raw federation (137 tools)	96.7%	$0.260	**90.6%**	$0.090
Compiled (rewrite+selection)	96.1%	$0.064	77.2%	$0.027
Compiled + consolidation	96.1%	**$0.052**	81.7%	**$0.022**
Consolidation, no selection	94.4%	$0.151	80.6%	$0.107
Cross-server consolidation	96.1%	$0.054	81.7%	$0.026

Table 1: Results over 60 tasks × 3 trials per condition per model. Sonnet CIs overlap fully between raw and the compiled conditions.

Four observations. First, for the stronger agent, *compilation is approximately free accuracy at one-fifth the cost*: all compiled conditions are statistically indistinguishable from the raw federation, and compiled consolidation wins both axes. Second, *compilation repairs cross-source arbitration*: on overlap-selection tasks the raw federation scored 90.0% with the stronger agent while every compiled condition scored 100% — pool-aware disambiguation rewriting outperforms the naive federation exactly where multiple vendors overlap. Third, and centrally, *the benefit is capability-gated*: the weaker agent reads the raw 25,000-token catalog competently (90.6%) but pays roughly nine points for compiled indirection (its errored-call rate rises from 5.3% to ∼24%), and its overlap-selection result inverts. Surface compression trades context for indirection, and the price of indirection falls with agent capability — approaching zero for frontier models. Deployments should select surfaces per agent model, which the toggle architecture makes a configuration choice. Fourth,

cross-server consolidation is safe but not yet differentiated: it ties compiled consolidation on both models; disambiguation rewriting appears to carry the arbitration load that explicit capability merging targets. A 30-task pilot preceding this study suggested a 100% ceiling for the no-selection condition; when doubled, that figure regressed to 94.4%, a reminder to distrust small-n ceilings.

Threats to validity. Small task count and trial count (wide intervals); judge circularity (a Claude model judging a Claude agent, mitigated by rubric-based judging and by human spot-checks); live-service drift between condition snapshots (mitigated by compiling all conditions from a single catalog snapshot); single agent model (the larger study adds model diversity).

7 Discussion

The compiler framing pays off in governance as much as performance. Because every transformation is an overlay over immutable originals, the system can show its work: consoles render exact diffs, facades display per-action provenance, and operator overrides compose with automated passes rather than fighting them. The same properties make LLM participation safe: models propose (rewrites, groupings, specification drafts, error fixes) while deterministic validators decide, and every acceptance is reviewable and reversible.

The runtime completes a loop that static compilation cannot: logged calls reveal which tools agents actually use and how they fail; failures become typed fix proposals; applied fixes recompile the surface. We believe this observe–optimize–recompile loop, rather than any single pass, is the durable contribution that provides maximum value.

References

[1] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

[2] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

[3] Y. Qin et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations (ICLR)*, 2024.

[4] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.

[5] M. Li, Y. Zhao, B. Yu, F. Song, H. Li, H. Yu, Z. Li, F. Huang, and Y. Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2023.

[6] C.-Y. Hsieh, S.-A. Chen, C.-L. Li, Y. Fujii, A. Ratner, C.-Y. Lee, R. Krishna, and T. Pfister. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*, 2023.

[7] S. Robertson and H. Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

[8] Anthropic. Introducing the Model Context Protocol. `https://modelcontextprotocol.io`, 2024.

[9] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations (ICLR)*, 2023.

[10] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 2nd edition, 2006.

[11] K. Pettis and R. C. Hansen. Profile guided code positioning. In *Programming Language Design and Implementation (PLDI)*, 1990.

[12] L. Zheng et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2023.

Availability

The engine is open source (Apache-2.0) at `https://github.com/toolc-dev/toolc` and on npm as `@toolc/*`; the hosted service is at `https://toolc.dev`.