AGENT-AS-A-TOOL FOR SERVICE PROVIDERS: CAPABILITY TRANSFER AND TOKEN SAVINGS AT SCALE

Anonymous authors
Paper under double-blind review

ABSTRACT

As LLM-based agents increasingly rely on external tools, should the tools for a dedicated service be exposed directly to the client agent's LLM (*Direct Tool Exposure*, DTE), or should they be wrapped behind a capable inner agent to which the client delegates tasks (*Agent-as-a-Tool*, AaT)? Existing studies predominantly take the client's perspective: given a fixed tool set registered with the client agent, they measure whether multi-agent orchestration improves the client's performance compared with a single LLM orchestrator. On the other hand, no existing study has quantified the benefit from the *provider's* perspective: what does a client agent gain when a service provider offers a capable agent wrapper as a single registered tool instead of directly exposing all low-level tools? In this paper, we present the first systematic comparison of DTE and AaT from the provider's perspective through a dedicated, controlled experiment. Our central finding is that AaT's benefits are predominantly explained by *capability transfer*: wrapping tools behind a service agent enables the client agent to draw on the service agent's capabilities, largely independent of the client LLM's own capabilities. In addition, AaT offers a token-efficiency advantage because DTE's token cost grows more sharply as the tool surface scales.

1 INTRODUCTION

Existing evaluations of multi-agent systems predominantly take the *client's* perspective, asking whether a tool-using agent benefits from orchestration mechanisms such as spawning sub-agents or interacting with other agents (Kim et al., 2025; Qin et al., 2023). The complementary question from a *service provider's* perspective is rarely asked: how does the provider's choice of service interface affect the quality received by downstream clients? A provider may expose the service's primitive tools directly (*Direct Tool Exposure*, DTE), requiring the client agent to plan and execute the tool calls itself, or wrap those tools behind a dedicated agent that accepts a natural-language (NL) request (*Agent-as-a-Tool*, AaT). For example, Playwright MCP (Microsoft, 2025) directly exposes browser-interaction tools such as clicking and typing, whereas Browser Use (Müller & Žunič, 2024) accepts NL requests through a dedicated browsing agent. Both interfaces support web-browsing tasks, but they place responsibility for planning and tool use on opposite sides of the service boundary.

Figure 1 illustrates this architectural difference. When the primitive tools, agent harness, and context-management policy are held fixed, the DTE client and the AaT inner service agent perform the same underlying tool-use loop. The distinguishing feature of AaT is the additional NL delegation boundary: instead of executing the task directly, the client communicates the task to the inner service agent, which then performs the tool-use steps. The central question is therefore what performance difference this NL delegation induces.To characterize this difference cleanly, we study a *single service-local* regime in which a task can be fully expressed in one request and completed using one provider's tools. The client need not inspect intermediate state or interleave calls to other services. The provider controls the service's APIs, schemas, policies, and recovery procedures and may encode them in the inner agent. Consequently, the client can issue one outer delegation call

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.

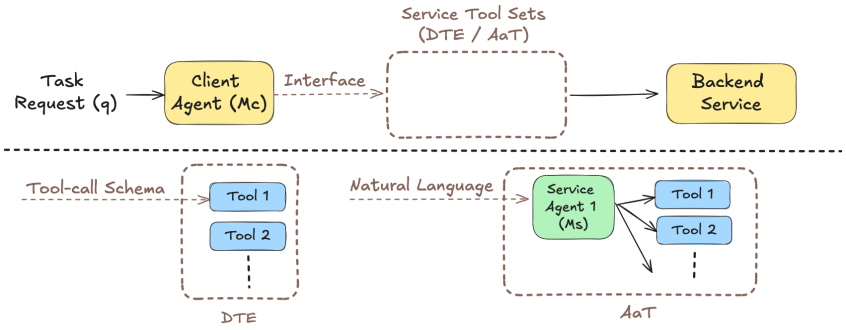

Figure 1: **Direct Tool Exposure (DTE) vs. Agent-as-a-Tool (AaT).** Top: a client agent M_c calling tools to complete a task q. Bottom-left (DTE): raw tools (for a single service) are directly exposed to the client agent and the client must plan, select tools, and recover from errors itself. Bottom-right (AaT): the tool system for a dedicated service is wrapped behind one callable agent (as a tool) instead — the client sends a single natural-language request, and an inner service agent M_s handles planning, tool selection, and error recovery.

while the inner agent takes multiple reasoning and tool-use steps. This deliberately scoped setting isolates the service-exposure boundary from iterative and cross-service orchestration[1].

Within this regime, we conduct a 41,250-run factorial experiment comparing AaT and DTE across five client models from three families and 250 tasks spanning five complexity tiers (tool-calling depth 2–20). The tasks run on a simulated document-management application with 21 primitive tools and deterministic gold states. Our *matchup classification* uses same-model controls to characterize the performance difference induced by NL delegation while holding the tool-executing model fixed. Stronger-inner comparisons then test whether capability transfer from a stronger service agent can offset any delegation-induced gap, while weaker-inner comparisons characterize possible degradation. Considered jointly, the results show that capability transfer is the primary opportunity through which AaT can benefit a provider's downstream clients. Specifically, this paper makes the following **contributions**:

1. We present the first controlled provider-side comparison of AaT and DTE, combining a formal framework, a matchup classification methodology, a factorial experimental design, and a reusable benchmark with deterministic gold states that we plan to open-source.
2. We show that AaT primarily transfers the inner model's capability: weak clients benefit substantially from a strong inner agent, whereas same-model controls provide no evidence of an intrinsic accuracy disadvantage from delegation.
3. We show that recent frontier LLMs are highly capable of selecting the correct tools from descriptions as the exposed tool surface grows. Consequently, AaT provides a purely mechanical token-saving advantage when services are effectively exposed as individual agents: it shields the client from primitive tool-schema growth while preserving accuracy over the tested ranges.

2 Related Work

Agent architectures and orchestration. Multi-agent LLM systems span diverse orchestration patterns (Guo et al., 2024; Tran et al., 2025): HuggingGPT (Shen et al., 2023) routes tasks to specialist models; Magentic-One (Fourney et al., 2024) formalizes an orchestrator-worker pattern; AutoGen (Wu et al., 2024), MetaGPT (Hong et al., 2024), and ChatDev (Qian et al., 2024) explore conversation-driven and role-based collaboration. However, empirical evidence increasingly challenges multi-agent complexity. MAST (Cemri et al., 2025) reports high failure rates across state-of-the-art frameworks, and budget-controlled studies question whether coordination helps once strong

[1]Without loss of generality, one can view the single service-local regime as an atomic level of tasks that require calling multiple services back-and-forth—which can be decomposed to multiple single service-local instances.

single-agent baselines are matched for compute (Tran & Kiela, 2026; Kim et al., 2025). Jwalapuram et al. (2026) diagnose the excess cost of automatically designed systems as "architectural bloat": generated topologies collapse into stochastic sampling. Their control complements ours—they fix the backbone and equalize the inference budget to separate compute from coordination, whereas we fix the client and vary the inner model to separate architecture from capability transfer. A concurrent theoretical account (Yu et al., 2026) formalizes why: under unbounded inter-agent communication bandwidth, a multi-agent system is provably equivalent to a single agent with the same context; any genuine multi-agent advantage must come from bandwidth-constrained, lossy communication rather than from task decomposition itself. Cognition (Cognition AI, 2025) now recommends defaulting to simpler architectures. Yet all evaluate from the *client's* perspective; our work asks the complementary *provider-side* question.

Tool use in LLMs. LLM tool use has evolved from self-taught invocation (Schick et al., 2023) and interleaved reasoning-acting (Yao et al., 2022) to systems scaling to thousands of APIs (Patil et al., 2024; Qin et al., 2023) with benchmarks standardizing evaluation (Patil et al., 2025). This line of work addresses *how well* models call tools, whereas our work addresses the orthogonal question of *how a provider should expose its tools (DTE vs. AaT)*. Prior work reports that tool-selection accuracy degrades as the candidate pool grows (Huang et al., 2024; Gan & Sun, 2025; Paramanayakam et al., 2025). Both Anthropic and OpenAI now advise limiting the number of tools in context for accuracy reasons (Anthropic, 2025; OpenAI, 2025b). Our tool-count scaling experiment (§5.3) therefore evaluates both accuracy and token cost as the exposed surface grows.

LLM routing and cascading. A separate line of work routes or escalates individual queries to a stronger model based on predicted difficulty or cost: RouteLLM (Ong et al.) trains a router offline on human votes over which of two models answered better, learning which queries the weak model handles without quality loss. More recent work formalizes the distinction between one-shot *routing* (choosing a single model upfront) and sequential *cascading* (escalating only after a cheaper model is tried) (Moslem & Kelleher, 2026). This literature shares our motivation of letting weak, cheap models borrow strong-model capability, but the mechanism differs fundamentally: routing and cascading select *which whole model* answers a query before or during execution, whereas AaT has one agent invoke *another whole agent*, mid-task, as a callable tool with its own multi-step tool-use loop. Under AaT, the client never predicts which model a query needs—it only determines which *tool* to call, with no router required.

The Agent-as-a-Tool pattern: adopted but unevaluated. Hierarchical delegation, as part of a broader shift toward compound AI systems (Zaharia et al., 2024), has roots in TaskMatrix.AI (Liang et al., 2024) and Chameleon (Lu et al., 2023), and was recently termed the "sub-agent-as-tools paradigm" (Ruan et al., 2026). In practice, Agent-as-a-Tool (AaT) pattern is also generally available and supported by different vendors. E.g., OpenAI's Agents SDK (OpenAI, 2025a) and AWS's Strands Agents SDK (Strands Agents, 2025) provide `Agent.as_tool()`, Google's Agent Development Kit (Google, 2025b) offers `AgentTool`, and the Model Context Protocol (MCP) (Anthropic, 2024) allows agent systems to be exposed as tool servers. Despite widespread recognition of the AaT framework, no prior work isolates AaT's effect on task success from the capability of the inner model, nor evaluates it from the provider's perspective. Systems employing AaT report aggregate performance but do not vary the inner model or client systematically, leaving it impossible to attribute success to delegation versus capability transfer. Our 5×5 cross-product with matchup classification provides the first such decomposition.

3 Framework and Research Questions

Task scope. We take the provider's lens and consider a provider-owned service for which its customer client's request q contains all task-specific context that the client must supply, and the service's tool set is sufficient to complete the task. The customer client need not inspect intermediate service state, revise q, or interleave calls to another service before completion. The provider controls the service interface and local execution guidance; we do not assume that the inner model has privileged knowledge of the task.

Architecture Definitions. Let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ be the set of n primitive tools in a domain system, M_c the client model, and M_s the server (inner) model. **Direct Tool Exposure (DTE):** the client M_c receives the full tool schema $\mathcal{T}$ and plans a sequence of tool calls $\langle t_{i_1}(\theta_1), t_{i_2}(\theta_2), \ldots \rangle$

directly to accomplish the task q. **Agent-as-a-Tool (AaT):** the client M_c receives a *single* tool $\hat{t}$ with interface $\hat{t}(q) \to r$, where q is a natural-language (NL) request and r is an NL response. In our operationalization, the client is able to invoke $\hat{t}$ once with the complete request; the server agent M_s may then take multiple internal turns and tool calls over $\mathcal{T}$ before returning r; recall Fig. 1 for the visual illustration.

Notation for systems and capability. We write DTE(M) for the DTE system whose client is model M, and AaT($M_c \to M_s$) for the AaT system in which client M_c delegates to inner agent M_s; AaT($\cdot \to M_s$) denotes the family of such systems over all clients. We write $M_1 \succ M_2$ when M_1 is more capable than M_2 under the fixed a-priori ranking of §4.3, and $M_1 \simeq M_2$ when that ranking does not separate them.

Analytical Cost Model. Consider a provider whose catalog $\mathcal{T}_{\text{all}}$ spans K services[2] and $N = |\mathcal{T}_{\text{all}}|$ tools in total. For a task requiring a ground-truth tool sequence of length k, the **DTE token cost** is:

$$C_{DTE} = \sum_{j=1}^{k'_c} \big(|P_c| + |\mathcal{T}_{\text{all,schema}}| + |H_j^c| + |O_j^c|\big) \tag{1}$$

where $k'_c \geq k$ includes retries, and P_c, $\mathcal{T}_{\text{all,schema}}$, H_j^c, O_j^c denote the client system prompt, the schema of all N exposed tools, and the client's conversation history and output at turn j. The **AaT token cost** splits into a thin client term and an inner term:

$$C_{AaT} = \underbrace{|P_c| + |\hat{t}_{\text{schema}}| + |q| + |r|}_{C_{\text{client}}} + \underbrace{\sum_{j=1}^{k'_s} \big(|P_s| + |\mathcal{T}_{\text{schema}}| + |H_j^s| + |O_j^s|\big)}_{C_{\text{inner}}} \tag{2}$$

where $\hat{t}_{\text{schema}}$ covers K agent-tools, one per service. C_{inner} is structurally analogous to C_{DTE} but *not* equal to it: the inner agent runs under its own prompt and its own trajectory, with its own retry count ($k'_s \neq k'_c$ in general), and sees only the n tools of the invoked service. Two asymmetries follow. First, DTE pays the full N-tool schema on *every* one of its k'_c turns, whereas AaT's client pays $|\hat{t}_{\text{schema}}| \ll |\mathcal{T}_{\text{all,schema}}|$ *once* under the scoped interface above, since each agent-tool exposes a single string parameter while $\mathcal{T}_{\text{all,schema}}$ describes all N tools and their typed arguments. Second, only C_{DTE} grows with the catalog size N; C_{inner} is pinned to n and C_{client} to K. Hence at a fixed small catalog ($N \approx n$) AaT is a strict addition—the extra inference layer is unrecovered overhead—but the differing N-dependence implies a crossover, which we locate empirically in §5.3.

Matchup Classification. Each AaT configuration simultaneously varies two factors: the *architecture* (NL delegation vs. direct tool access) and the *model (M_c or M_s) executing the tool calls*. To disentangle these sources of variation, we classify every AaT-vs-DTE comparison using a fixed a-priori capability ranking of models (see §4.3): **same-model** matchups ($AaT(M_c \to M_c)$ vs. $DTE(M_c)$) hold capability constant, estimating the net effect of the delegation boundary (NL serialization, formatting divergence, and the single outer-call constraint); **stronger-inner** matchups ($M_s > M_c$) test whether access to a more capable inner agent overcomes that boundary; and **weaker-inner** matchups ($M_s < M_c$) characterize the degradation boundary. The effect of capability-transfer can thus be revealed by combining the observations of the above matchups jointly.

Research Questions (RQ). The cost model reveals an inherent tension: AaT simplifies the client interface at the expense of an additional NL delegation layer. Within the service-local scope above, we ask the following questions: **RQ1 (Task Success):** what is AaT's impact in terms of success rate, and what underlying mechanism drives such impact? **RQ2 (Token Cost):** under what regime does delegation call demonstrates pure token overhead? Can AaT save tokens when the tool surface grows?

[2] See §1 for definition of "service".

4 EXPERIMENTAL SETUP

A fair comparison of DTE and AaT requires a testbed in which performance differences can be attributed to the architecture rather than to model strength, environment noise, or the task domain. Existing tool-use benchmarks are designed to score a single agent's tool calls, not to wrap the same tool set behind a provider-side agent and re-score it: BFCL (Patil et al., 2025) and ToolBench (Qin et al., 2023) lack persistent, verifiable end states and long-horizon depth, while τ-bench (Yao et al., 2024) places a stochastic simulated user directly on the delegation boundary we aim to isolate (Appendix A.1). We therefore construct a simulated, controlled testbed.

4.1 SIMULATED CONTROLLED TESTBED: DESIGN REQUIREMENTS AND INSTANTIATION

Design requirements. Concretely, the testbed must: **(R1)** score deterministic persistent state; **(R2)** vary tool-call depth while keeping individual steps expertise-free; **(R3)** inject recoverable errors that require an alternative tool; **(R4)** provide a clean base tool surface that can be expanded with confusable neighboring services; **(R5)** avoid floor and ceiling effects across models; and **(R6)** reset exactly and cheaply at experimental scale. Appendix A.1 lists full details why existing benchmarks do not suffice.

Controlled document-management testbed. We implement 21 primitive tools spanning document management, search, text editing, and media. Each run's local directory state reduces to a hash (R1, R6), while familiar, expertise-free operations compose into gold sequences of 2–20 calls (R2, R5). Text tools fail deterministically on protected image, equation, and diagram regions, forcing recovery through the corresponding media tools (R3). The base tool names are orthogonal for the capability experiment; the scaling experiment adds confusable neighboring services, making ambiguity the measured phenomenon rather than a confound (R4). Documents are natively linguistic, avoiding domain-intrinsic serialization loss at the NL delegation boundary, and the synthetic `.unidoc` format combines familiar markdown, LaTeX, and mermaid primitives without exposing memorized workflows. Full design rationale and tool catalog are in Appendix A.

Table 1: Task complexity tiers. 50 tasks per tier, 250 total. Error triggers cause recoverable tool failures (e.g., protected-region violations).

Tier	Description	Depth	Errors	N
L1	Multi-tool, single doc	2–4	No	50
L2	Multi-document ops	3–6	No	50
L3	Error-prone ops	3–8	Yes	50
L4	Deep multi-document	$\sim$20	No	50
L5	Deep error-prone	$\sim$20	Yes	50

4.2 ARCHITECTURE INSTANTIATION AND TASK GENERATION

Architecture instantiation: As illustrated in Figure 1, the client receives all 21 tool schemas and orchestrates them directly under DTE. For AaT, the client sends one complete NL request to a `document_agent`, which handles planning and execution internally. This transaction-style contract is compatible with tool APIs, MCP servers, and `Agent.as_tool()` interfaces, but it is our experimental operationalization rather than a requirement of those protocols. Asking the outer client to one call isolates the service-exposure boundary: allowing it to inspect intermediate outputs and delegate again would also measure client-side replanning, feedback handling, and additional inference. We therefore distinguish this setting from iterative client-side orchestration (Wu et al., 2024; Fourney et al., 2024; Hong et al., 2024).

Task generation and complexity tiers: All 250 tasks are produced by a deterministic, template-based generator (Appendix A.3). Gold final states are computed by executing gold tool sequences on initial states using the actual tool implementations, and each task is validated by comparing a whitespace-normalized SHA-256 hash (National Institute of Standards and Technology, 2015) of every output document against the corresponding gold final state. Table 1 describes the five tiers with varying tool sequence length and error density—the two cost drivers identified by Eqs. 1–2.

Table 2: **Calibrated success rates (%) across all tiers.** L1–L3 and L4–L5 are tier-group averages. $\Delta_{\max}$ is the largest AaT improvement over DTE on L4–L5 for each client; bold marks the best configuration within each tier group. Strong inner models substantially narrow capability gaps across clients, with the largest gains for the weakest clients, while same-model pairings show no general architectural accuracy advantage. Inferential details are reported in Table 9; per-tier results appear in Appendices G.

	DTE (%)		AaT→Opus (%)		AaT→Sonnet (%)		
Client	**L1–L3**	**L4–L5**	**L1–L3**	**L4–L5**	**L1–L3**	**L4–L5**	$\Delta_{\max}$
Qwen3 32B	61.6	11.0	93.6	**94.7**	**93.8**	70.0	+83.7
DeepSeek-V3.2	78.7	33.3	95.3	**95.0**	**96.0**	81.7	+61.7
Haiku	91.1	47.7	96.2	**93.7**	**97.1**	77.0	+46.0
Sonnet	96.0	83.7	96.7	**93.3**	**96.9**	77.7	+9.6
Opus	**96.7**	**96.3**	**96.7**	95.3	93.6	78.7	−1.0

4.3 Models, Matchups, and Scaling

We adopt a fixed a-priori ranking based on general model capabilities (Arena Intelligence, 2026)—Opus 4.6 > Sonnet 4.6 (Anthropic, 2026) > Haiku 4.5 > DeepSeek-V3.2 (DeepSeek-AI et al., 2024; 2025) > Qwen3 32B (Yang et al., 2025)[3]—established before any experiments to avoid circularity.

Tool-count scaling setup. To test RQ2 when the number of tools and services increase, we add task-irrelevant distractors from 9 semantically related services and evaluate 21, 50, 100, and 200 total tools. DTE exposes all tools in one list; AaT exposes one agent-tool per service (10 services in total). Therefore, the expected task trajectory from the dataset will always invoke the document (service) agent for AaT construct, which always sees the same 21 real tools. The distractor design removes simple shortcuts: DTE tool names omit service prefixes and return plausible responses, while all AaT agent-tools share the same single-parameter (NL delegation) schema. We test Opus, Sonnet, and Haiku in same-model pairings on L2 and L4, holding capability fixed while tool count varies. The 21-tool DTE anchor was rerun with the scaling prompt and supports 21–200 inference. Full catalog, anchor details, and run breakdown are in Appendix H; success uses the metric in §4.4.

4.4 Evaluation

Task success. The primary metric is **calibrated success**: a whitespace-normalized hash comparison of the final document state against the gold state, augmented by an LLM-judge (Opus 4.6, 3-sample majority vote) that re-evaluates hash-match failures. The judge reclaims runs where the agent produced semantically correct output that was penalized by rigid formatting criteria arising from unforeseen task-description ambiguity. This calibration is important for fair AaT evaluation: delegation may introduce formatting divergence because a single task description (in the original input, not after delegation) may admit multiple correct final states (e.g., both with and without trailing newlines). Details and validation examples are in Appendix B.

Supplementary metrics: token cost and error recovery. We record total input and output tokens per run, broken down by agent role. For AaT, *total tokens* (inner-agent trajectory + NL delegation) are compared with DTE tokens.[4] On tasks with error triggers (L3 and L5), we report the *error recovery rate*: whether the agent ultimately succeeded despite encountering tool-execution failures.

5 Experiments and Results

Each (task, configuration) pair runs 3 times (temperature 0.3), and both architectures share the same per-tier tool-call budget (Appendix A.4). The full experiment spans 55 system configs × 250 tasks × 3 reps = **41,250 runs**. The primary analysis focuses on 15 practitioner-relevant configs (5 DTE base-

[3]For brevity, we will refer to the Claude models by product name (Opus, Sonnet and Haiku) alone onward. We still use the full name for other models because they are only one of their families.

[4]We do not report latency: all models use shared cloud API endpoints, so wall-clock time reflects server-side load and throttling rather than architectural differences. Isolating AaT's extra serial call requires controlled serving infrastructure (future work).

lines and 10 AaT configurations pairing each client with Opus or Sonnet inner—a provider deploying its strongest model, §4.3). All success rates use the calibrated metric (§4.4).

Statistical methodology. We first average the three repetitions of each task within each configuration, then compare configurations on the resulting paired task means. Primary tests for RQ1 use the directional alternative AaT > DTE implied by the hypothesis and capability ranking fixed before the experiments; other difference tests are two-sided. We report percentage-point (pp) differences with 95% task-bootstrap confidence intervals (20,000 joint resamples of tasks), alongside paired-sample Cohen's d_z for continuity with prior work. Benjamini–Hochberg (BH) FDR correction (Benjamini & Hochberg, 1995) is applied separately within the seven primary transfer tests, and the nine exploratory scaling head-to-head tests. Equivalence claims use two one-sided tests (TOST) with an explicitly post-hoc ±5pp practical margin and a stricter ±3pp sensitivity check; see Appendix F.

5.1 AaT: A Capability Transfer Mechanism with Negligible Delegation Cost

Pairing a weaker client with a strong inner model under AaT dramatically improves task success through capability transfer, with negligible delegation cost. Table 2 presents calibrated success rates across all five tiers. With Opus (the strongest) as inner, the L4–L5 gains are enormous for weak clients: Qwen3 32B jumps from 11.0% to 94.7% [5], DeepSeek-V3.2 from 33.3% to 95.0% [6], and Haiku from 47.7% to 93.7% [7]. All seven stronger-inner contrasts are BH-significant (Table 9).

On simple and moderate tasks (L1–L3), AaT with a strong inner model drives all clients to near-ceiling performance (93–97%), erasing the large DTE capability gaps visible between the weaker models and Opus. At higher complexity (L4–L5), the inner model's dominance becomes apparent. With Opus as the inner model, the five observed client means lie between 93.3% and 95.3%, a descriptive 2.0pp spread despite their DTE baselines spanning 11.0–96.3% [8]. The weakest clients see the largest gains, while the strongest client, Opus, loses 1.0pp, as shown in Fig. 2. With Sonnet as inner, performance is lower (70.0–81.7% on L4–L5) and more client-dependent (11.7pp spread), but the inner Sonnet model still transfers substantial capability to weaker clients; also see Fig. 2. This is a clear evidence that the inner model is the dominant performance factor in this setting.

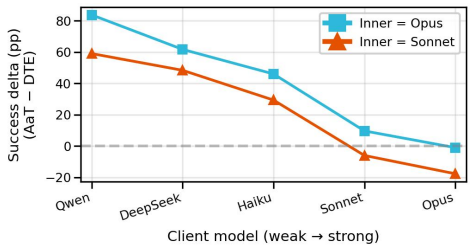

Figure 2: **Strong inner models help weaker clients most.** For L4–L5 tasks, the plot shows how much calibrated success changes when each client uses AaT with Opus or Sonnet as the inner model instead of DTE. Clients are ordered from weaker to stronger; values above zero favor AaT. The benefit decreases as client capability rises.

Same-model architectural control. Same-model comparisons hold model capability constant, isolating the net cost of the delegation boundary. In Table 2's Opus row, AaT (Opus→Opus) achieves 95.3% on L4–L5 versus 96.3% under DTE, a −1.0pp difference [9]. In the Sonnet row, AaT (Sonnet→Sonnet) reaches 77.7% versus 83.7% under DTE, a −6.0pp estimate [10]. Thus the delegation boundary is practically small for Opus at the stated 5pp tolerance, while the Sonnet estimate is less precise and potentially larger. Appendix E shows the detailed failure analysis that the observed Opus L4 gap is concentrated in *one single* ambiguous formatting template—a special failure mode of Opus rather than a general delegation gap. Therefore, this further supports that AaT is

[5] +83.7pp, 95% task-bootstrap CI [77.3, 89.3].

[6] +61.7pp, task-bootstrap CI [53.3, 70.0].

[7] +46.0pp, task-bootstrap CI [37.7, 54.3].

[8] The 2.0pp spread is measured under the calibrated metric; under the raw hash the spread is 8.7pp, since instruction-silent formatting divergence varies with how each client phrases its delegated request (Appendix B.4). The capability-transfer improvements for weaker clients remain large under either metric.

[9] $d_z = -0.05$; 95% task-bootstrap CI $[-3.7, 1.3]$pp. TOST supports equivalence under the post-hoc ±5pp margin (90% paired-t CI $[-3.15, 1.15]$pp, $p = 0.001$), but narrowly not under ±3pp ($p = 0.063$).

[10] $d_z = -0.15$; 95% task-bootstrap CI $[-12.7, 0.7]$pp that does not establish either a two-sided difference ($p = 0.080$) or ±5pp equivalence ($p = 0.615$).

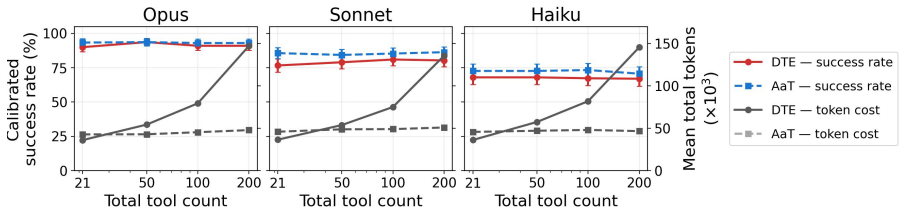

Figure 3: **Tool-count scaling.** Each panel compares same-model DTE and AaT for one client model on L2 and L4 tasks. Colored lines show calibrated success (left axis), and gray lines show mean token use (right axis). As the number of tools grows, we detect no accuracy trend for either approach, but token use rises sharply for DTE and changes little for AaT. The 21-tool AaT point is shown only for reference; growth comparisons use 50–200 tools. Error bars are descriptive; statistical results are in Appendix H.

Client	Mean sim.	Median	% identical
Qwen3 32B	0.99	1.00	66%
DeepSeek-V3.2	0.95	1.00	68%
Haiku	0.83	0.91	34%
Sonnet	0.78	0.84	14%
Opus	0.95	1.00	42%

Table 3: Delegation fidelity: similarity between the client's transmitted request and the original instruction, over the primary strong-inner AaT runs ($n \approx 1{,}500$ per client). "% identical": fraction of runs where the transmitted request is byte-identical to the instruction.

a capability transfer mechanism for strong models, while the delegation gap has negligible impact on task success.

5.2 Delegation Fidelity by Client

The expected mechanism for capability transfer is that clients relay the task faithfully into the inner agent, so the inner agent receives essentially the original instruction. We measure this delegation fidelity directly. For every AaT run, we log the client's transmitted request and compare it to the original task instruction via `SequenceMatcher`(Python Software Foundation, 2026) similarity; Table 3 reports the per-client distribution across the primary strong-inner configurations.

The two weakest clients relay near-verbatim (mean similarity 0.95–0.99, roughly two-thirds byte-identical), consistent with a simple mechanism for their large gains: the stronger inner agent receives nearly the raw task. Fidelity does not decrease monotonically with capability—Sonnet restructures the request most (mean similarity 0.78, mean length ratio 1.36, i.e. it tends to expand the request), and Opus relays faithfully again—more capable models does not necessarily imply more delegation fidelity. Notably, the observed transfer pattern persists with Sonnet's heavier restructuring: success under a Sonnet client still varies with the inner model's capability (§5.1). This association suggests AaT's tolerance to moderate paraphrasing, but does not isolate fidelity as a causal mediator.

5.3 AaT can save tokens when the number of tools and service increases

Recent frontier LLMs have robust tool calling performance when the number of registered tools increase. Contrary to the reported tool-calling performance degradation as candidate pools grow in Huang et al. (2024); Gan & Sun (2025) and the vendor guidance placing the onset at 30–50 tools (Anthropic, 2025), none of the tool-calling performance in our experiment shows a tread in degrading accuracy for DTE over its same-prompt 21–200 series.

Token cost grows rapidly for DTE and less for AaT. Over the fully comparable 50–200 (raw) tools window, DTE's mean token cost grows 2.5–2.7×, whereas AaT changes by -1% to $+11\%$. The mechanism is architectural: DTE retransmits the full tool schema every turn, while AaT's inner agent always sees the same 21 real tools and the client's surface grows only *sub-linearly* in total tool count

N (the client sees $K \in \{4, 8, 10\}$ agent-tools as N grows $\{50, 100, 200\}$, as grouped by de-coupled service). This same-model cost gap is structural, making it a second argument for wrapping tool-rich services behind AaT with a performant model. Exact growth rates, token breakdowns, and the cost premium at 200 tools are in Appendix H. Note that Fig. 3 shows uncached counts; prompt caching reduces but does not remove DTE's schema-dependent growth, while context-window occupancy is unaffected (Appendix D.2).

Fixed-surface overhead and error recovery. At 21 tools, same-model AaT stays within ∼0–15% of DTE's token cost on L1–L3 but adds ∼7–19% on L4 and ∼23–27% on L5. Client delegation is only 4.3–5.3% of total AaT tokens, that means for recent frontier models, the impact of delegation gap (recall §5.2) can propagate its impact into the inner agent's task solving trajectories. Error recovery follows the same capability-transfer pattern as success, where the weakest agents benefits the most from delegating the task to a strong inner service agent. Full results are in Appendices D and C.

6 Discussion and Conclusion

Limitations. Our controlled testbed covers one synthetic document-management domain, so the quantitative results may not generalize to other tool surfaces, failure modes, or production workloads. The single-call design applies to service-local tasks that can be specified upfront, and is meant to construct a clean variable control for experimentation; iterative or cross-service orchestration may have different accuracy and cost tradeoffs. The tool count scaling study covers three Claude models and at most 200 tools, and its intervals permit moderate accuracy changes, so it does not establish non-degradation. Finally, Sonnet did not consistently follow the intended single-call protocol in the scaling study - although this has no impact on demonstrating capability transfer, strict token estimates for single-call Sonnet regime will require a hard-enforced rerun, which is minor for the content of this paper.

Recommendations for service design and service owners. As MCP (Anthropic, 2024) and A2A (Google, 2025a) commoditize the AaT pattern, our findings yield the following guidelines for providers (e.g., Software-as-a-service (SaaS) companies) when considering to expose tool-rich services to empower agentic access from their customers.

First, wrapping a strong agent as a tool sells capability access, not an intrinsic architectural accuracy advantage. AaT's large gains arise predominantly from capability transfer from the inner model. That means, if the service provider serves their best model and agentic harness for their service under AaT, their customers' client agent will always enjoy the best possible service quality, regardless of the model on their customers' end. One step further, the AaT construct leaves the following interesting future work to explore—whether using the most capable generalist frontier LLM (e.g., the ones from the most famous frontier labs), or training a dedicated service model specifically for the service domain serves AaT better? Whether optimizing the delegation fidelity is worth to be considered an important training objective for client LLM for better AaTutilization? We leave these topics out of the scope of this paper and as a future work.

Second, service provider surfacing AaT can save their clients token cost. This is because the inner agent's tool surface is decoupled from how many other services are exposed to the customer's client. Note that this does not imply that adding an agent layer is generally economical no matter how AaT is instrumented—AaT's token saving advantage assumes that (1) the service owner provides the best possible inner agents and (2) services registered under the client are decoupled. Both assumptions are reasonable in practice, because (1) the service owner should know their service best and (2) companies typically do not share service data across different parties. On the other hand, a complimentary study on a client-side orchestration regime by (Jwalapuram et al., 2026) shows that blindly adding extra topology only contributes deliberation without improving accuracy, inflating total cost by up to 10×. Their results therefore identify a different boundary: unnecessary client orchestration can waste compute, whereas our provider-side schema isolation can save it.

7 AI USE STATEMENT

Paper polishing. AI was used to help polish the paper writing. However, all contents are proof read by human authors.

Related work retrieval. AI (deep search) was used to help source related works. However, all comments on the mentioned related works are manually checked.

Research idea and execution. AI was used to collaborate with the authors in articulating research ideas and implementing experiment code. During this process, AI is always an executor of the human authors input. No Deep Research(Team, 2026) style practice has been exercised for this paper.

Synthetic dataset generation. As part of the code instrumentation, the synthetic dataset is completely generated by AI with human verification.

8 REPRODUCIBILITY STATEMENT

Our study requires no model training or local accelerator compute: every agent is an off-the-shelf model queried through a commercial inference API, so the experiments can be reproduced with API access alone. The testbed, its 21 primitive tools, and the full tool catalog are specified in §4.1 and Appendix A. All 250 tasks are produced by a deterministic template-based generator with a fixed random seed (seed=42; Appendix A.3); gold final states are obtained by executing the gold tool sequences with the same tool implementations the agents use, and each task is validated by hash comparison against its gold state. Every run executes in an isolated sandbox reset to the task's initial state, so the environment itself is deterministic. The client and inner-agent system prompts, the single-call delegation protocol, turn limits, and per-tier tool-call budgets are given in Appendix A.4 and the subsection that follows it, and the distractor-service catalog for the scaling experiment in Appendix H. Each (task, configuration) pair is run three times at temperature 0.3 (§5). The calibrated success metric, including the LLM-judge rubric and 3-sample majority vote, is described in §4.4 and Appendix B, with a sensitivity analysis against purely deterministic metrics in Appendix B.4; all statistical procedures (task-level pairing, bootstrap resampling, BH correction, and TOST margins) are detailed in Appendix F. Upon publication, we will release under a license permitting research use the complete source code (testbed, tools, architectures, experiment runner, and analysis scripts), the task generator and generated tasks, and the raw per-run results, including full conversation traces, final document states, and LLM-judge verdicts, so that every reported number can be recomputed without re-querying the models. Residual variation stems from provider-side sampling and possible model-version updates, which we mitigate with repeated runs and task-level confidence intervals.