WHAT DRIVES LLM TOOL SELECTION IN MCP? A FACTOR-LEVEL ANALYSIS THROUGH CONTROLLED METADATA INTERVENTIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

In Model Context Protocol (MCP) systems, LLMs select tools from metadata rather than from their implementations. Names and descriptions summarize what a tool does, while the parameter schema specifies the fields a tool accepts and how they are organized. Our analysis of metadata for 367,174 MCP tools reveals substantial schema variation among functionally similar tools. Many of these tools share names and similar descriptions but expose different parameter schemas. This raises a key question: can parameter-schema differences influence tool selection before any call is made? We study this question through controlled metadata interventions across five LLMs and five tasks. Each comparison changes exactly one metadata factor (server name, tool name, description, or parameter schema) while keeping the others fixed. We balance tool positions and use conditional logistic regression to estimate position-adjusted effects. We find that: (1) parameter-schema changes produce the largest and most consistent selection shifts among the tested interventions. Models tend to favor separate top-level parameters over grouped or composite forms. (2) Descriptions that explain parameters also tend to increase selection, while name effects are weaker and less consistent. (3) Targeted comparisons show that schema effects depend on the specific parameter names and on how the request represents the same information. When only one schema satisfies an explicit request requirement, all five models select the compatible tool despite position preferences. These results show that parameter schemas influence not only how tools are called, but also whether they are selected. For MCP developers and evaluators, schema design and tool ordering can therefore affect which implementation is invoked even when functionality is unchanged.

1 INTRODUCTION

The Model Context Protocol (MCP) lets an application connect to tools from many independently developed MCP servers (Model Context Protocol, 2026a;b). As tool ecosystems grow, an LLM may need to choose among tools with overlapping functionality. The model does not observe a tool's implementation directly; it sees an interface representation consisting of a server name, a tool name, a description, and a parameter schema (Figure 1b). Which tool the model selects determines which implementation runs, so the choice matters for the reliability of tool-using systems, for developers whose servers compete to be used, and for anyone who might edit tool metadata to steer the model.

These metadata fields are designed for different purposes. Names and descriptions explicitly summarize what a tool does, whereas the parameter schema is primarily designed to specify how the tool should be invoked. The MCP ecosystem shows why this distinction deserves attention. In metadata for 367,174 MCP tools (Li & Gao, 2026), tools with the same function often share a name and a similar description but differ widely in their parameters: the 169 records named `send_email` use 78 distinct parameter sets (Figure 1a). From their names and descriptions, these tools are easy to recognize as doing the same thing; they differ mainly in details intended for invocation, such as whether the recipient field is called `to` or `recipient_id` (Appendix A.2). This raises the possibility that such details already influence which tool is selected, before any call is made. We ask whether they do, and how their influence compares with that of names and descriptions.

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.

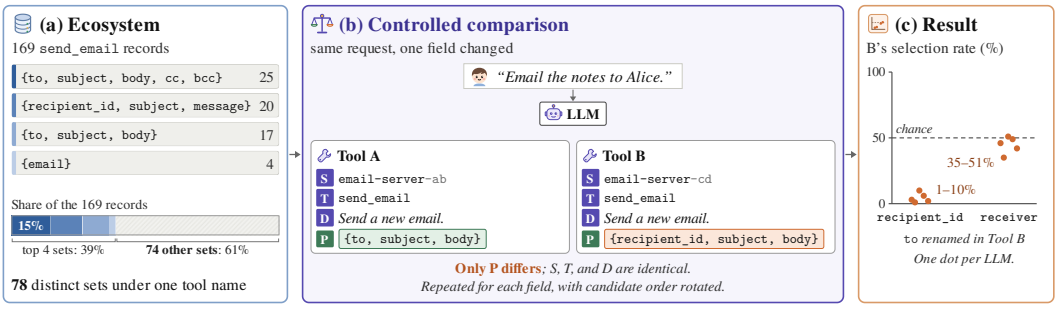


Figure 1: Overview of the study (full MCP workflow in Appendix Figure 10). (a) Tools that share a name often take different parameters: the 169 `send_email` records in our dataset use 78 distinct parameter sets. (b) In each comparison, the LLM receives the same request and candidate tools that differ in exactly one metadata field: server name (S), tool name (T), description (D), or parameters (P). The host joins S and T into a single name field, and a two-letter server suffix only distinguishes the candidates. Here, Tool B renames `to` to `recipient_id`. (c) Selection rate of Tool B in two-tool comparisons, with one dot per LLM. Renaming `to` to `recipient_id` lowers it to 1–10%, whereas renaming it to `receiver` keeps it at 35–51%, close to chance.

Prior work has shown that tool selection can be biased (Blankenstein et al., 2026) and that edits to tool descriptions can shift which tool a model prefers (Faghih et al., 2025; Wang et al., 2026c); related work has also documented quality problems in MCP tool descriptions (Hasan et al., 2026). The role of parameter schemas has received substantially less attention, particularly in controlled comparisons that isolate schema changes from changes to names and descriptions.

Ecosystem data alone cannot answer this question, because tools that differ in their schemas usually differ in other metadata as well. We therefore intervene directly. For five common tasks, we construct tool variants that differ in exactly one metadata field, using forms observed in the ecosystem, and present them together as candidates for the same user requests. Because candidate order can itself affect selection, we balance tool positions and adjust for position with a conditional logit model (Train, 2003).

In this paper, we make three contributions.

First, we introduce a controlled framework for studying MCP tool selection. It spans four metadata fields, five tasks, five LLMs, and candidate sets of two, three, and five tools. Each comparison changes a single field, reuses the same requests, and balances candidate positions, isolating the effect of each tested edit.

Second, we find that schema edits produce the largest and most consistent shifts in selection among the tested edits. We distinguish three kinds of schema edits: renaming parameters, grouping parameters into a single object, and splitting a composite input into separate fields. Models tend to favor separate top-level fields, such as `owner` and `repo` rather than a single `owner/repo` string, and naming effects depend strongly on the particular names being compared. Descriptions that explain parameters also tend to increase selection, whereas the effects of server and tool names are weaker and vary across models. When candidates differ only in their names, models often pick the first-listed tool.

Third, we examine when and why schema effects arise. Renaming `to` to `recipient_id` reduces a tool's selection rate to 1–10% across models, whereas renaming it to `receiver`, a word absent from the requests, leaves the tool competitive (35–51%). The effect therefore does not require word overlap with the request, and is consistent with `recipient_id` signaling an identifier rather than an email address. The preference for separate `owner` and `repo` fields weakens for every model, and reverses for one, when the same request gives the repository as a single `owner/repo` path. Finally, when a request asks for a CC recipient that only one schema can accept, all five models select that tool, even when it is listed second. Together, these results indicate that models use schema information to judge whether a tool fits a request during selection, rather than only to format arguments after selection.

These findings have practical consequences. For server developers, schema design affects not only how a tool is called but whether it is called at all. For host designers and evaluators, the strength of order effects means that tool-selection outcomes should be measured under varied candidate orders.

2 Related Work

Tool selection. Most tool-use benchmarks evaluate whether a model selects and calls the correct tool (Huang et al., 2024; Patil et al., 2025), including in MCP settings (Luo et al., 2025; Wang et al., 2026a) and after retrieval narrows the candidate set (Gan & Sun, 2025). Robustness studies perturb queries, toolkits, or tool documentation and measure the resulting loss in accuracy (Ye et al., 2024; Rabinovich & Tavor, 2025). In these settings, one candidate is correct. We instead study selection among functionally similar tools, where no candidate is wrong, and the question is what drives the model's choice.

Metadata effects on selection. Selection among equivalent tools is biased (Blankenstein et al., 2026), and edits to tool descriptions can increase a tool's usage by more than an order of magnitude (Faghih et al., 2025). Adversarial work exploits the same channel through optimized names and descriptions (Sneh et al., 2025; Wang et al., 2026c), injected tool documents (Shi et al., 2025), and poisoned MCP metadata (Wang et al., 2026b; Li & Gao, 2026). These studies focus on names and descriptions; parameter schemas are rarely varied in isolation. Anand & Chattaraj (2026) include a parameter trap that adds an unsatisfiable required argument, testing whether agents avoid unusable tools. In our comparisons, every candidate can serve the request; schemas differ only in naming, structuring, or decomposition, so a selection shift reflects preference rather than feasibility.

Tool documentation. Tool documentation alone can enable zero-shot tool use (Hsieh et al., 2023), and concise rewrites of tool instructions improve agent performance (Yuan et al., 2025). For MCP, tool descriptions often contain quality defects, and augmenting them involves trade-offs between task performance and execution cost (Hasan et al., 2026). These works mainly examine how documentation affects tool use once a tool is chosen; we study how tool metadata, from descriptions to parameter schemas, shapes which tool is chosen in the first place.

Order effects and choice modeling. LLMs are sensitive to option order in multiple-choice questions (Zheng et al., 2024; Pezeshkpour & Hruschka, 2024), and AI shopping agents show strong position biases along with sensitivity to product descriptions (Allouah et al., 2026; Kumar & Lakkaraju, 2024). We therefore balance tool positions and include them as covariates in a conditional logit model (Train, 2003), which estimates both metadata effects net of position and the position effects themselves.

3 Modeling Tool Selection

We treat each trial as a choice among candidate tools and estimate how changing one metadata factor shifts that choice, separately from the effect of where each tool appears.

3.1 Setup

Given a task request q and an ordered list of candidate tools $C = (c_1, \ldots, c_m)$ with $m \in \{2, 3, 5\}$, we represent each tool by the four metadata factors the model observes:

$$c_i = (S_i, T_i, D_i, P_i), \tag{1}$$

namely the server name, tool name, description, and parameter schema. Tool c_i appears at position i, and $Y \in \{1, \ldots, m\}$ denotes the position of the selected tool.

Metadata variants. For each factor $f \in \{S, T, D, P\}$, we define two variants, v_{high} and v_{low}, and let $x_{i,f} \in \{0, 1\}$ indicate whether tool c_i uses v_{high} ($x_{i,f} = 1$) or v_{low} ($x_{i,f} = 0$). For server names, tool names, and parameter schemas, v_{high} is the more common form within the task cluster and v_{low} a less common one (Appendix A); parameter alternatives differ in naming, structuring, or decomposition (Table 1). For descriptions, v_{low} briefly states the tool's function, and v_{high} adds

Table 1: Parameter variations across the five tasks.

Dimension	Task	v_{high}	v_{low}
Naming	Send Email	`to`	`recipient_id`
	Get Weather	`location`	`input`
Structuring	Create Issue	`(owner, repo, ...)`	`params_dict`
	Place Order	`(customer_id, ...)`	`create_order_requests`
Decomposition	Create Pull Request	`(owner, repo)`	`repository`

Note. v_{high} is the more common form in each task cluster. Naming changes parameter names; structuring groups separately declared parameters into a single container; decomposition splits a composite parameter into multiple parameters. Full parameter definitions are provided in Table 3 in the appendix.

explanations of parameters or return values, following prior work on description quality (Hasan et al., 2026). A positive effect of v_{high} thus indicates a preference for the more common form for S, T, and P, and for the more complete description for D.

Controlled comparisons. Each comparison varies a single factor f across the m candidates and keeps the other three fixed, so candidates differ only in $x_{i,f}$ and position.

3.2 Conditional Logit Model

We model the selected position with a conditional logit model (Train, 2003):

$$\Pr(Y = i \mid q, C) = \frac{\exp(\beta_f x_{i,f} + \gamma_i)}{\sum_{j=1}^{m} \exp(\beta_f x_{j,f} + \gamma_j)}, \tag{2}$$

where β_f measures the effect of using v_{high} rather than v_{low}, and γ_i measures the effect of appearing at position i rather than last ($\gamma_m = 0$). Because the other three factors are fixed across candidates, their contributions are constant within a trial and cancel in Eq. (2). Estimated jointly, β_f captures the metadata effect with position held fixed, and γ_i captures the position effect with metadata held fixed; a positive β_f thus reflects a preference for v_{high} beyond any position preference.

Estimation. Each trial forms one choice group. For each factor, LLM, task, and candidate-set size, we estimate (β_f, γ) by maximizing

$$\mathcal{L}(\beta_f, \gamma) = \sum_{r=1}^{R} \log \Pr(Y = Y_r \mid q_r, C_r), \tag{3}$$

where Y_r, q_r, and C_r are the selected position, request, and candidate list in trial r, and R is the number of valid trials, i.e., trials in which the model calls a candidate tool (Section 4). We fit each model by maximum likelihood and report 95% Wald confidence intervals, with standard errors from the inverse observed information. When a fit fails because choices are completely or quasi-completely separated, no finite estimate exists, and we mark the estimate as *sep*.

3.3 Selection Rates

To describe observed choices without model assumptions, we also report selection rates. Let $H_r = x_{Y_r,f}$ indicate whether the tool selected in trial r uses v_{high}. The v_{high} selection rate and the position-k selection rate are

$$\text{SR}_f^{\text{high}} = \frac{1}{R} \sum_{r=1}^{R} H_r, \qquad \text{SR}_k^{\text{pos}} = \frac{1}{R} \sum_{r=1}^{R} \mathbf{1}[Y_r = k], \tag{4}$$

computed separately for each candidate composition (e.g., 1H+2L; Section 4), with R counting the trials in that composition. Under uniform choice, the baseline for $\text{SR}_f^{\text{high}}$ is the fraction of candidates using v_{high}, and that for SR_k^{pos} is $1/m$. Selection rates summarize observed choices, whereas the conditional-logit coefficients separate metadata effects from position effects.

4 EXPERIMENTAL SETUP

Tasks and queries. We study five tasks: Send Email, Get Weather, Create Issue, Place Order, and Create Pull Request, which cover the parameter naming, structuring, and decomposition differences in Table 1. The metadata variants are listed in Appendix B. For each task, we use 50 natural-language queries generated with GPT-5.6 from the task's core inputs (e.g., recipient, subject, and body for Send Email) and manually checked. Queries vary in wording and inputs and never mention server or tool names. The same queries are reused across all factors and assignments. For Create Pull Request, the queries form 25 matched pairs that give the same request but specify the owner and repository either separately (*split*) or as a combined `owner/repo` path (*path*); we analyze the two formats separately.

Models. We evaluate GPT-5.4 mini (OpenAI, 2026) (GPT), Claude Sonnet 4.6 (Anthropic, 2026) (Claude), Gemini 3.5 Flash (Google DeepMind, 2026) (Gemini), DeepSeek V4 Pro (DeepSeek-AI, 2026) (DeepSeek), and Qwen3.7-Plus (Qwen Team, 2026) (Qwen). Temperature and other sampling settings are left at their API defaults. Thinking is disabled where supported; for Gemini, we set the thinking level to minimal. Tool choice is set to *auto* with parallel tool calls disabled, and we record the called tool as the model's selection. All experiments were run in August and September 2026.

Candidate tools. Each comparison presents $m \in \{2, 3, 5\}$ candidate tools. The target factor takes v_{high} or v_{low} in each candidate, and the other three factors are fixed at their v_{low} forms. Beyond the target factor, candidates differ only in a random two-letter server suffix (e.g., `-xl`). As in MCP hosts, the server and tool names are joined into a single name field (e.g., `message-server-xl__send_gmail`). Using H and L to denote v_{high} and v_{low}, the candidate compositions are 1H+1L for two tools; 1H+2L and 2H+1L for three tools; and 2H+3L and 3H+2L for five tools.

Assignments and presentation order. An assignment specifies which candidates use v_{high}. We enumerate all assignments with one H among two candidates, one or two H among three, and two or three H among five, yielding 2, 6, and 20 assignments, so each candidate uses v_{high} equally often. Presentation order is a cyclic rotation of the candidates whose offset depends on the query and assignment, so each candidate appears at each position equally often; the conditional logit model adjusts for any remaining position imbalance. Each query is run once per factor and assignment, giving 400, 1,200, and 4,000 trials per task–model pair for two, three, and five tools, with trial order shuffled using a fixed random seed. For Create Pull Request, trials are split equally between the two query formats. We exclude trials in which the request failed or no candidate tool was called; in every task–model configuration, these account for at most 1.3% of trials.

5 RESULTS

5.1 METADATA EFFECTS ON TOOL SELECTION

Table 2 reports conditional-logit coefficients for three-tool comparisons across five tasks and five models, and Appendix Figure 7 shows the corresponding selection rates for Create Issue.

Parameter schemas have the largest and most consistent effects. Of the 28 numerical P estimates, 23 have 95% confidence intervals that exclude zero, and 22 of these are positive. The two remaining P cells are separated because Gemini and Qwen select a v_{high} tool in every Create Issue trial. Among coefficients whose intervals exclude zero, P is the largest in 19 of the 30 task–model conditions, and its median magnitude (3.82) is more than twice that of S (1.48). Several large estimates with wide intervals, such as GPT's 16.23 in Create Issue, reflect near-separation rather than weak effects. For the two naming tasks, Send Email and Get Weather, all five models have positive P estimates, favoring the common names `to` and `location`; Section 5.4 shows that such naming effects depend on the particular names being compared.

Models favor separate top-level fields. When parameters are grouped into a single object, models tend to prefer the variant with separate top-level parameters. In Create Issue, v_{high} tools are

Table 2: Conditional-logit estimates of metadata effects on tool selection with three candidate tools.

Task	Model	S	T	D	P
Send Email	GPT-5.4 mini	2.39	1.98	1.37	**4.13**
	Claude Sonnet 4.6	1.74†	1.24†	0.00†	**4.56**
	Gemini 3.5 Flash	1.54	1.62	0.55	**7.88**
	DeepSeek V4 Pro	0.01†	−0.83†	2.49	**3.03**
	Qwen3.7-Plus	5.64†‡	−0.30†	5.64†‡	11.77†
Get Weather	GPT-5.4 mini	−0.89	0.38†	1.02	**4.44**
	Claude Sonnet 4.6	2.15	0.65†	1.73	**7.87**
	Gemini 3.5 Flash	2.26	0.14†	1.72	**5.22**
	DeepSeek V4 Pro	1.01	−0.26†	**2.49**	1.78
	Qwen3.7-Plus	0.00†‡	−7.34†	1.80	**2.16**
Create Issue	GPT-5.4 mini	**7.12**	0.01†	2.35	16.23†
	Claude Sonnet 4.6	0.20†	0.16†	1.72	**6.32**
	Gemini 3.5 Flash	−2.29	0.23†	2.36	**sep***
	DeepSeek V4 Pro	2.81	0.12†	1.71	**4.84**
	Qwen3.7-Plus	sep	0.20†	4.74	**sep***
Place Order	GPT-5.4 mini	0.43	0.54	1.35	**4.30**
	Claude Sonnet 4.6	7.29†	0.01†	10.65†	19.41†
	Gemini 3.5 Flash	0.03†	−11.32†	1.98	**2.51**
	DeepSeek V4 Pro	−0.51†	−0.47†	0.01†	−0.33†
	Qwen3.7-Plus	0.03†‡	0.03†‡	0.03†‡	11.81†
Create Pull Request (*split*)	GPT-5.4 mini	−0.23†	sep	0.15†	**3.82**
	Claude Sonnet 4.6	1.48	0.53†	−0.47†	**5.47**
	Gemini 3.5 Flash	0.59	0.39†	1.43	**3.07**
	DeepSeek V4 Pro	1.14	0.78	0.40†	**1.67**
	Qwen3.7-Plus	1.42	0.42†	1.16	**6.05**
Create Pull Request (*path*)	GPT-5.4 mini	1.27	−0.49†	0.50	**1.95**
	Claude Sonnet 4.6	**3.08**	0.89	−0.07†	2.78
	Gemini 3.5 Flash	0.85	0.65	1.13	**1.48**
	DeepSeek V4 Pro	**1.23**	0.28†	0.72	0.51
	Qwen3.7-Plus	2.25	8.73†	**2.71**	−1.01

Note. Entries are conditional-logit coefficients (β_f) for v_{high} vs. v_{low}, adjusted for position. S: server name; T: tool name; D: description; P: parameter schema. †95% confidence interval includes zero; large estimates with † (e.g., 16.23) reflect near-separation rather than weak effects. sep: separation, so no finite estimate exists; *the model selected a v_{high} tool in every trial. ‡Qwen's choices in these comparisons are determined by position, so its metadata effects are not identified and these estimates should not be interpreted as effects. For Create Pull Request, *split* and *path* queries give the owner and repository separately or as a combined `owner/repo` path.

selected in 93.3%–100% of trials across models and compositions, well above the corresponding baselines. In Place Order, P estimates are positive for four models, while DeepSeek shows no clear preference ($\beta_P = -0.33$, interval including zero). For decomposition, all five models favor separate `owner` and `repo` parameters over a single `repository` field when the query gives the owner and repository separately, and four still do so when it gives a combined `owner/repo` path. Section 5.4 examines how this preference changes with the request format.

Descriptions that explain parameters increase selection. Of the 30 D estimates, 21 have intervals that exclude zero, and all 21 are positive. This holds for every model in Get Weather and Create Issue, and for Gemini in every task and query format. In Create Issue, v_{high} descriptions are also generally selected above their composition-specific baselines (Appendix Figure 7). Explaining parameters thus increases selection even when the brief description already states the tool's function.

Name effects are weaker, and position often decides. Tool names rarely matter: only 6 of 29 numerical T estimates have intervals that exclude zero. Server-name effects are more frequent (19 of 29), but they are smaller than schema effects and reverse for two task–model pairs (GPT in Get Weather and Gemini in Create Issue). When candidates differ only in names, position often determines the choice. Under tool-name comparisons in Create Issue, v_{high} selection rates stay near the uniform-choice baselines, whereas GPT, Claude, DeepSeek, and Qwen select the first-listed tool in 79%–96% of trials (Appendix Figure 7). Under server-name comparisons, Claude, DeepSeek, and Qwen show similar position preferences, whereas GPT favors v_{high} and Gemini favors v_{low}.

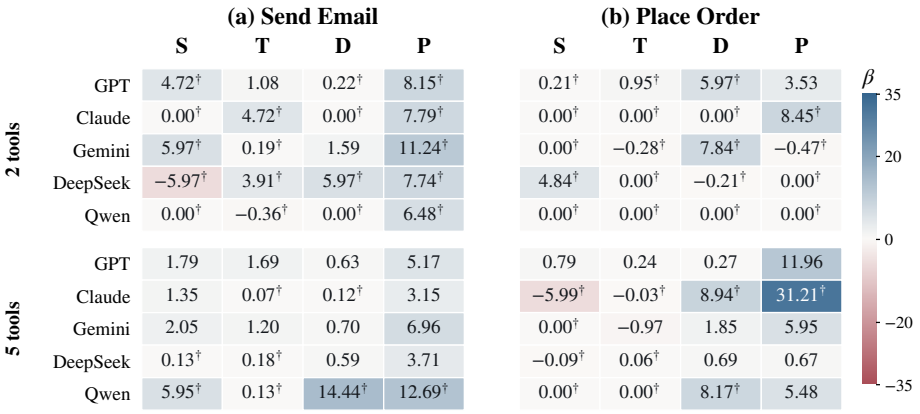


Figure 2: Metadata effects on tool selection with two and five candidate tools for (a) Send Email and (b) Place Order. Cells show conditional-logit coefficients (β) for v_{high} vs. v_{low}, adjusted for presentation position. S: server name; T: tool name; D: description; P: parameter schema. Both panels use the same color scale. † indicates that the 95% confidence interval includes zero.

5.2 Candidate-Set Size and Position Effects

Figure 2 compares coefficients with two and five candidate tools for Send Email and Place Order (see also Appendices C.2–C.3), and Figure 3 shows the two-tool selection rates for Place Order.

Parameter effects persist with five tools. With five candidates, P estimates are positive for all ten task–model pairs, and eight have intervals that exclude zero. The other two, Qwen in Send Email (12.69) and Claude in Place Order (31.21), are large estimates that reflect near-separation. In seven of the ten pairs, P is the largest coefficient whose interval excludes zero. The schema preference found with three tools therefore does not depend on a particular candidate-set size.

With two tools, position dominates. Most two-tool estimates have intervals that include zero, and several are exactly zero. Figure 3 explains why: under server- and tool-name comparisons in Place Order, all five models pick the first-listed tool in 92%–100% of trials, while v_{high} rates stay near 50%. Claude always picks the first tool under S, T, and D, and Qwen under all four factors, so Qwen's near-zero coefficients reflect separation by position rather than absent metadata effects.

Only strong metadata preferences overcome position. Parameter comparisons are the main exception: GPT selects the tool with separate top-level parameters in 97% of trials but the first-listed tool in only 51%, and Claude selects that tool in 76%. Gemini instead follows descriptions (81% v_{high}) rather than schemas (44%). A single metadata difference thus outweighs position only when the preference is strong, so selection outcomes should be measured under varied candidate orders.

5.3 Joint Effects of Metadata Factors

The comparisons above vary one field at a time. To test whether schema effects persist when fields vary together, we run a two-tool factorial experiment on Send Email in which the two candidates differ in all four fields. For each field, one candidate uses v_{high} and the other v_{low}; which candidate receives each variant is crossed across the four fields and presentation order, yielding $2^5 = 32$ conditions. Each of the 50 queries is run once per condition, giving 1,600 trials per model, and we estimate the effects of S, T, D, P, and position jointly (Figure 4). The parameter contrast here concerns naming (Table 1).

Parameter effects remain the largest. P has the largest coefficient for four of the five models. For GPT, Gemini, and DeepSeek, these coefficients (3.92, 4.16, and 4.50) have intervals that exclude zero; Claude's estimates are large for most factors but imprecise. Qwen is the exception, with

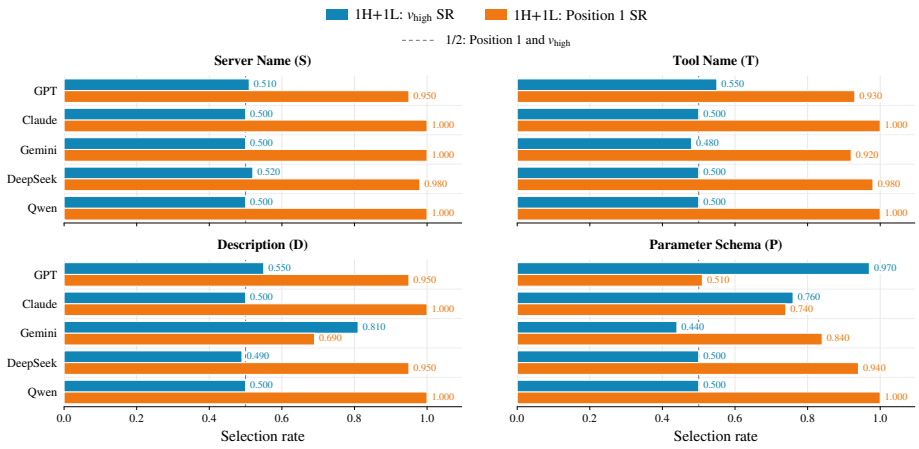


Figure 3: Two-tool selection rates for Place Order. Each comparison contains one v_{high} and one v_{low}, with 100 valid trials per model and factor. Blue bars show the rate of selecting the v_{high} candidate; orange bars show the rate of selecting the first tool. The dashed line marks the uniform-choice baseline of 0.5 for both rates.

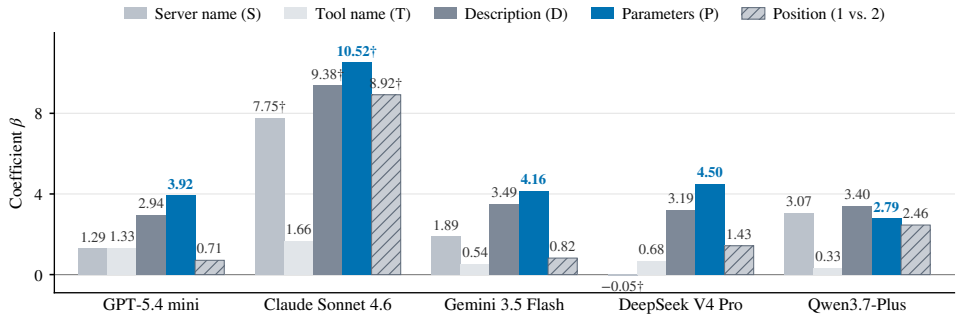


Figure 4: Joint metadata and position effects in the two-tool factorial experiment (Send Email). Bars show conditional-logit coefficients (β) for v_{high} vs. v_{low} in each field and for position 1 vs. 2, estimated jointly; parameter effects in blue. †: 95% confidence interval includes zero.

its largest coefficient for descriptions ($\beta_D = 3.40$), although its parameter effect remains positive ($\beta_P = 2.79$). Descriptions rank second for GPT, Gemini, and DeepSeek, whereas tool-name effects stay small (at most 1.66), consistent with Section 5.1.

Joint differences can outweigh position. For the four models with precise estimates, the position effect (0.71–2.46) is smaller than the parameter effect. Unlike the single-field comparisons in Section 5.2, where position often decides, choices among candidates that differ in several fields depend more on metadata than on position. Parameter-name effects thus persist when multiple metadata fields vary together.

5.4 When and Why Do Schema Effects Arise?

We now examine when and why schema effects arise, using two-tool comparisons that vary parameter names and task requirements (Figure 5; Appendices C.4–C.5) and the matched query formats of Create Pull Request (Table 2).

Parameter names signal what input a tool expects. Replacing `to` with `recipient_id` lowers the renamed tool's selection rate to 1.0%–10.0% across all five models, whereas replacing `to` with `receiver` keeps it at 35.2%–51.0%, and replacing `body` with `text` yields near-balanced rates of 45.5%–54.6% (Figure 5a). Because `receiver` never appears in the queries, a renamed tool can

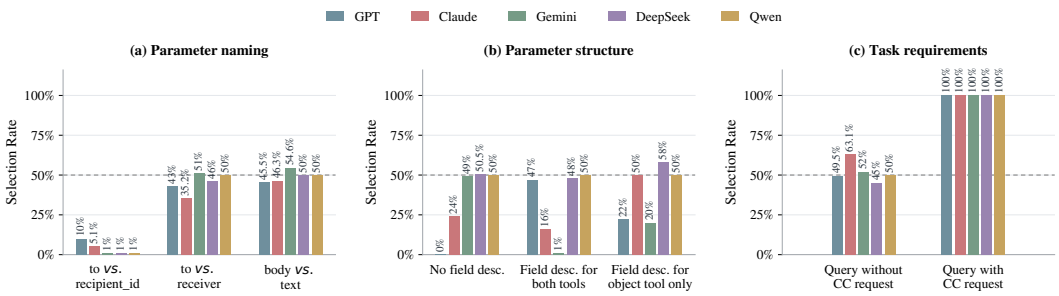


Figure 5: Two-tool selection rates under changes to parameter names, field descriptions, and task requirements. (a) Send Email: rate of selecting the tool with the renamed parameter; other parameters are unchanged. (b) Place Order: rate of selecting the tool with a single object parameter rather than separate named parameters. (c) Send Email: rate of selecting the tool with an additional optional `cc` parameter, with and without an explicit CC request. Dashed lines mark the 50% baseline.

remain competitive without word overlap with the request. The drop for `recipient_id` is instead consistent with the name signaling an identifier rather than an email address: `receiver` and `text` preserve the roles of recipient and message content, whereas `recipient_id` suggests a different kind of input. Identical field descriptions remove this drop (Appendix C.4).

Schema preferences depend on how the request expresses inputs. The matched split and path queries for Create Pull Request differ only in whether the owner and repository are given separately or as a combined `owner/repo` path. Moving from split to path queries lowers the P coefficient for every model: from 3.82 to 1.95 for GPT, 5.47 to 2.78 for Claude, 3.07 to 1.48 for Gemini, and 1.67 to 0.51 for DeepSeek. Qwen's coefficient reverses, from 6.05 to -1.01, so it favors the single `repository` field when the request supplies a single path. Field descriptions on structured parameters, by contrast, have no consistent effect: adding them to the object parameter in Place Order raises the object variant's selection rate for some models and lowers it for others (Figure 5b).

Task requirements can override position. In Send Email, one tool adds only an optional `cc` field to the shared `to`, `subject`, and `body`. Without a CC request, it is selected in 45.0%–63.1% of trials and the first-listed tool in 81.0%–100% (Figure 5c). With a CC request, all five models select it and satisfy the requirement in every valid trial, even when it is listed second. If it also renames `to` to `recipient_id`, however, three models select it in only 53.0%–67.4% of these trials (Appendix C.5).

Together, these results indicate that models use schema information to judge whether a tool fits a request during selection, rather than only to format arguments after selection.

6 Conclusion

We studied what drives LLM tool selection among functionally similar MCP tools through controlled metadata interventions across five LLMs and five tasks. After adjusting for position, parameter schemas had the largest and most consistent effects, favoring separate top-level fields across candidate-set sizes and under joint variation. Descriptions that explain parameters also increased selection, whereas names mattered less and often yielded to position. Schema effects depend on what parameter names signal and on how the request expresses its inputs, and explicit task requirements can override position. Models thus use schemas to judge whether a tool fits a request during selection: schema design affects whether a tool is called at all, and tool-selection outcomes should be measured under varied candidate orders.

Limitations. Our findings are limited to the tested models, tasks, metadata variants, and candidate-set sizes, and to single-turn, LLM-generated English requests without tool execution. We measure selection behavior rather than decision processes, so our interpretation of naming effects (e.g., `recipient_id` signaling an identifier) remains behavioral.

ETHICS STATEMENT

This work studies how tool metadata affects LLM tool selection and involves no human subjects or personal data. The ecosystem analysis uses publicly available MCP tool metadata from Li & Gao (2026), and all queries were generated for this study with fictitious recipients and inputs (e.g., `example.com` addresses). Because our findings show that metadata changes can shift which tool is selected, they could inform attempts to steer model choices; we report them to help developers and host designers anticipate such effects and evaluate tool selection under varied metadata and candidate orders.

REPRODUCIBILITY STATEMENT

Section 3 specifies the conditional logit model and selection rates, and Section 4 describes the tasks, queries, models, API settings, candidate compositions, and trial schedules, including the dates of all API calls. Appendix B lists the full metadata variants and parameter schemas. The supplementary material includes all queries and the request and analysis scripts, with the fixed random seeds used to generate trial schedules. Because the evaluated models are accessed through APIs that may change over time, exact results may differ in later runs.

AI USE STATEMENT

In this work, we used generative AI tools to identify functionally similar tools in the ecosystem data (Cursor with Grok 4.6; Appendix A.1), to generate the natural-language queries for each task (GPT-5.6; Section 4), to help implement the experiment and analysis code, and to assist in summarizing results during writing. We did not use generative AI tools for any other purpose requiring disclosure, such as generating research ideas or formulating hypotheses. Additionally, we used generative AI tools to edit the paper for readability, identify relevant literature, and revise figures. We have reviewed all AI-assisted work: generated queries were checked for correctness and for the absence of server or tool names, AI-assisted code was tested by the authors, reported statistics were verified against the analysis outputs, and all cited works were checked against their original sources. The five LLMs evaluated in our experiments are the subjects of study rather than tools used to produce this work. We take responsibility for the final content of this work.

REFERENCES

Amine Allouah, Omar Besbes, Josué D Figueroa, Yash Kanoria, and Akshit Kumar. What is your ai agent buying? evaluation, biases, model dependence, & emerging implications of agentic e-commerce. In *Proceedings of the ACM Web Conference 2026*, pp. 8697–8700, 2026.

Atul Anand and Sourav Chattaraj. Diagnosing tool-selection reasoning in llm agents with canary tools. *arXiv preprint arXiv:2608.04719*, 2026.

Anthropic. System card: Claude Sonnet 4.6. https://www.anthropic.com/claude-sonnet-4-6-system-card, 2026. Accessed September 2026.

Thierry Blankenstein, Jialin Yu, Zixuan Li, Vassilis Plachouras, Sunando Sengupta, Philip Torr, Yarin Gal, Alasdair Paren, and Adel Bibi. Biasbusters: Uncovering and mitigating tool selection bias in large language models. In *International Conference on Learning Representations*, volume 2026, pp. 102460–102489, 2026.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. https://huggingface.co/deepseek-ai/DeepSeek-V4-Pro, 2026. Technical report.

Kazem Faghih, Wenxiao Wang, Yize Cheng, Siddhant Bharti, Gaurang Sriramanan, Sriram Balasubramanian, Parsa Hosseini, and Soheil Feizi. Tool preferences in agentic llms are unreliable. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20954–20969, 2025.

Tiantian Gan and Qiyao Sun. Rag-mcp: Mitigating prompt bloat in llm tool selection via retrieval-augmented generation. *arXiv preprint arXiv:2505.03275*, 2025.

Google DeepMind. Gemini 3.5 Flash model card. https://deepmind.google/models/model-cards/gemini-3-5-flash/, 2026. Accessed September 2026.

Mohammed Mehedi Hasan, Hao Li, Gopi Krishnan Rajbahadur, Bram Adams, and Ahmed E Hassan. Model context protocol (mcp) tool descriptions are smelly! towards improving ai agent efficiency with augmented mcp tool descriptions. *arXiv preprint arXiv:2602.14878*, 2026.

Cheng-Yu Hsieh, Si-An Chen, Chun-Liang Li, Yasuhisa Fujii, Alexander Ratner, Chen-Yu Lee, Ranjay Krishna, and Tomas Pfister. Tool documentation enables zero-shot tool-usage with large language models. *arXiv preprint arXiv:2308.00675*, 2023.

Yue Huang, Jiawen Shi, Yuan Li, Chenrui Fan, Siyuan Wu, Qihui Zhang, Yixin Liu, Pan Zhou, Yao Wan, Neil Gong, et al. Metatool benchmark for large language models: Deciding whether to use tools and which to use. In *International Conference on Learning Representations*, volume 2024, pp. 42978–43007, 2024.

Aounon Kumar and Himabindu Lakkaraju. Manipulating large language models to increase product visibility. *arXiv preprint arXiv:2404.07981*, 2024.

Xiaofan Li and Xing Gao. A first look at the security issues in the model context protocol ecosystem. In *2026 56th Annual IEEE International Conference on Dependable Systems and Networks (DSN)*, pp. 379–392. IEEE, 2026.

Ziyang Luo, Zhiqi Shen, Wenzhuo Yang, Zirui Zhao, Prathyusha Jwalapuram, Amrita Saha, Doyen Sahoo, Silvio Savarese, Caiming Xiong, and Junnan Li. Mcp-universe: Benchmarking large language models with real-world model context protocol servers. *arXiv preprint arXiv:2508.14704*, 2025.

Model Context Protocol. Build an MCP client. https://modelcontextprotocol.io/docs/develop/build-client, 2026a. Accessed September 19, 2026.

Model Context Protocol. Tools. https://modelcontextprotocol.io/specification/2026-07-28/server/tools, 2026b. MCP specification, version 2026-07-28. Accessed September 19, 2026.

OpenAI. Introducing GPT-5.4 mini and nano. https://openai.com/index/introducing-gpt-5-4-mini-and-nano/, 2026. Accessed September 2026.

Shishir G Patil, Huanzhi Mao, Fanjia Yan, Charlie Cheng-Jie Ji, Vishnu Suresh, Ion Stoica, and Joseph E Gonzalez. The berkeley function calling leaderboard (bfcl): From tool use to agentic evaluation of large language models. In *Forty-second International Conference on Machine Learning*, 2025.

Pouya Pezeshkpour and Estevam Hruschka. Large language models sensitivity to the order of options in multiple-choice questions. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pp. 2006–2017, 2024.

Qwen Team. Qwen3.7-Plus: Multimodal agent intelligence. https://qwen.ai/blog?id=qwen3.7-plus, 2026. Accessed September 2026.

Ella Rabinovich and Ateret Anaby Tavor. On the robustness of agentic function calling. In *Proceedings of the 5th Workshop on Trustworthy NLP (TrustNLP 2025)*, pp. 298–304, 2025.

Jiawen Shi, Zenghui Yuan, Guiyao Tie, Pan Zhou, Neil Zhenqiang Gong, and Lichao Sun. Prompt injection attack to tool selection in llm agents. *arXiv preprint arXiv:2504.19793*, 2025.

Jonathan Sneh, Ruomei Yan, Jialin Yu, Philip Torr, Yarin Gal, Sunando Sengupta, Eric Sommerlade, Alasdair Paren, and Adel Bibi. Tooltweak: An attack on tool selection in llm-based agents. *arXiv preprint arXiv:2510.02554*, 2025.

Kenneth E. Train. *Discrete Choice Methods with Simulation*. Cambridge University Press, 2003.

Zhenting Wang, Qi Chang, Hemani Patel, Shashank Biju, Cheng-En Wu, Quan Liu, Aolin Ding, Alireza Rezazadeh, Ankit Parag Shah, Yujia Bao, et al. Mcp-bench: Benchmarking tool-using llm agents with complex real-world tasks via mcp servers. In *International Conference on Learning Representations*, volume 2026, pp. 97377–97425, 2026a.

Zhiqiang Wang, Yichao Gao, Yanting Wang, Suyuan Liu, Haifeng Sun, Haoran Cheng, Guanquan Shi, Haohua Du, and Xiangyang Li. Mcptox: A benchmark for tool poisoning on real-world mcp servers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 35811–35819, 2026b.

Zihan Wang, Rui Zhang, Yu Liu, Wenshu Fan, Wenbo Jiang, Qingchuan Zhao, Hongwei Li, and Guowen Xu. Mpma: Preference manipulation attack against model context protocol. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 35838–35846, 2026c.

Junjie Ye, Yilong Wu, Songyang Gao, Caishuang Huang, Sixian Li, Guanyu Li, Xiaoran Fan, Qi Zhang, Tao Gui, and Xuan-Jing Huang. Rotbench: A multi-level benchmark for evaluating the robustness of large language models in tool learning. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 313–333, 2024.

Siyu Yuan, Kaitao Song, Jiangjie Chen, Xu Tan, Yongliang Shen, Kan Ren, Dongsheng Li, and Deqing Yang. Easytool: Enhancing llm-based agents with concise tool instruction. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 951–972, 2025.

Chujie Zheng, Hao Zhou, Fandong Meng, Jie Zhou, and Minlie Huang. Large language models are not robust multiple choice selectors. In *International Conference on Learning Representations*, volume 2024, pp. 19426–19454, 2024.

A More Details on Tasks

This appendix describes how we derive task clusters from the MCP ecosystem (§A.1), illustrates the variation within one cluster (§A.2), and details the queries and the targeted comparisons in Section 5.4 (§A.3–§A.4). The metadata variants and their coding are given in Appendix B.

A.1 Ecosystem Data and Task Clusters

Metadata records. We use the metadata of 367,174 MCP tools collected by Li & Gao (2026) from five MCP registries: mcp.so, MCP Market, MCP Store, Pulse MCP, and Smithery. Each record contains the hosting platform, the server identifier, the tool name, the tool docstring, and the names of its parameters and return values. We map these fields to the four metadata factors: the server identifier (e.g., `cristip73/mcp-email-server`) to the server name S, the tool name (e.g., `send_email`) to T, the docstring to the description D, and the parameter list (e.g., `to`, `subject`) to the parameter schema P.

Task clusters. As in Blankenstein et al. (2026), we group functionally similar tools into task clusters. We use an LLM agent (Cursor with Grok 4.6 as the backbone model) to identify, based on tool descriptions, the tools that perform a given task, such as tools that can send an email. Because the data aggregate five registries, the same tool can appear more than once; we remove these duplicates, as well as tools whose parameter list or docstring is missing or empty. When counting name forms, we normalize tool names for case, underscores, hyphens, and spaces (e.g., `send_email`, `sendEmail`, and `send-email` map to the same name).

From naming to structure and decomposition. Inspecting the resulting clusters, we found that tools sharing a name and a similar description often differ in their parameters, as in the Send Email cluster (Appendix A.2). The most direct difference is parameter naming, such as `to` versus `recipient_id`. Starting from naming, we identified two further ways in which the same inputs can be specified: grouping separately declared parameters into a single container (structuring) and splitting a composite input into multiple parameters (decomposition). Table 1 summarizes the three dimensions.

Task selection. For each dimension, we select representative task clusters that contain many tools and exhibit the corresponding variation: Send Email and Get Weather for naming, Create Issue and Place Order for structuring, and Create Pull Request for decomposition. After filtering, these clusters contain 288 (Send Email), 639 (Get Weather), 247 (Create Issue), 109 (Place Order), and 83 (Create Pull Request) records. Within each cluster, we count the forms of server names, tool names, and parameter schemas to identify the more and less common variants (Appendix B).

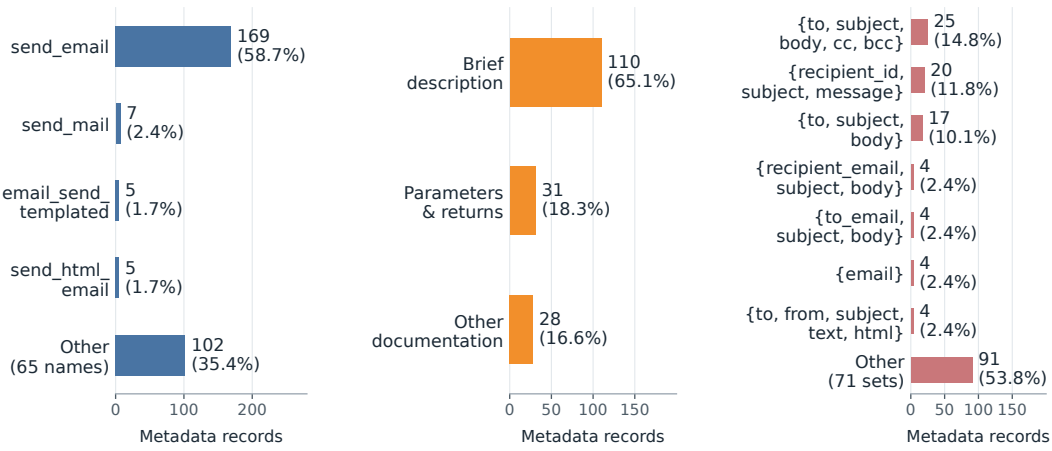
A.2 Variation Within the Send Email Cluster

Figure 6 shows the distributions of tool names, description content, and parameter sets in the Send Email cluster. Among the 288 deduplicated records in the cluster, `send_email` is the most common normalized name (169 records, 58.7%), while 65 other names account for 35.4%. Among the 169 `send_email` records, most descriptions are brief (65.1%), and 18.3% explain parameters and return values. The same 169 records use 78 distinct parameter sets: the most common, {`to`, `subject`, `body`, `cc`, `bcc`}, covers only 14.8% of records, and 71 low-frequency sets together account for 53.8%. Tools that share a name and a similar description therefore differ widely in how their inputs are specified, which motivates isolating the parameter schema from the other fields.

A.3 Queries

For each task, we prompted GPT-5.6 with the task's core inputs to generate 50 queries that vary in wording and values, and the authors checked each query for correctness and for the absence of server or tool names. All recipients and other personal details are fictitious. A representative Send Email query is:

> *Email kevin@example.com with the subject 'Delivery Update' and tell him that the delivery is expected to arrive on Thursday.*



(a) Tool name (b) Description (c) Parameters

Figure 6: Tool names, description composition, and parameter sets in the Send Email cluster. *Notes.* Panel (a) uses all 288 deduplicated metadata records; tool names are normalized for case, underscores, hyphens, and spaces. Panels (b) and (c) use the 169 records mapped to `send_email`. Parameter sets ignore order and preserve exact names. Low-frequency names and parameter sets remain counted in *Other*.

For Create Pull Request, the queries are built from 25 base tasks, each specifying a repository, source and target branches, and a body, using five sentence templates. Each base task yields a matched pair that differs only in how the repository is referenced; branches, body, wording, tool assignments, presentation order, and shuffle seeds are otherwise identical. The repositories are illustrative and are never modified. For example:

> ***Split:*** Create a new pull request in the GitHub repository 'web-dashboard' under the organization 'cedar-labs' to merge branch 'fix/login-button' into branch 'main'. Set its body to 'Fix the unresponsive login button after valid credentials are entered.'.
>
> ***Path:*** Create a new pull request in the GitHub repository 'cedar-labs/web-dashboard' to merge branch 'fix/login-button' into branch 'main'. Set its body to 'Fix the unresponsive login button after valid credentials are entered.'.

A.4 Targeted Comparisons

The comparisons in Section 5.4 use the two-tool design of Section 4, with the two candidates differing only in the property under study.

Parameter names. Starting from the Send Email tool with {`to`, `subject`, `body`}, we rename a single parameter in one candidate: `to` to `recipient_id`, `to` to `receiver`, or `body` to `text`. All other parameters and fields are identical. We use the 50 Send Email queries with 200 trials per model per comparison.

Field descriptions. In Place Order, we compare the tool with separate top-level parameters (`customer_id`, `item_id`, `quantity`) against the tool that groups the same fields under a single `create_order_requests` object, under three conditions: no field descriptions, field descriptions on both tools, and field descriptions on the object tool only. Field descriptions are attached to the three leaf fields ("Numeric customer ID", "Numeric item ID", and "Quantity of items"); the wrapper object has no description, and both tools keep the tool-level description "Place a new order". We use the 50 Place Order queries, and each query is run once with each tool in the first position, giving 100 trials per condition and model.

Task requirements. In Send Email, the base tool exposes {`to`, `subject`, `body`} and the extended tool adds only an optional `cc` field. Both tools are named `send_email`, share the description "Send a new email." without field descriptions, reject undeclared fields, and differ otherwise only in their two-letter server suffixes, which are counterbalanced across queries. We use 50 paired

queries: in the *unspecified* condition, the query asks to send an email without mentioning CC; in the *required* condition, the same request, with the same recipient, additionally asks to CC a specified address. The paired queries are built on the Send Email queries of Appendix A.3 with the same generation and checking procedure. Each query is run with the extended tool in each of the two positions, giving 100 trials per condition and 200 per model. Selection rates are computed over trials with exactly one call to a candidate tool. A trial satisfies the CC requirement if the model selects the extended tool with schema-valid arguments, the correct recipient, and the requested CC address.

B METADATA VARIANTS AND CODING

This appendix lists the two variants of each metadata factor for all five tasks. Table 3 summarizes the names and parameter forms; Appendix B.3 gives the full descriptions and parameter schemas.

B.1 CODING CRITERIA

Names and parameter forms. Within each task cluster (Appendix A.1), we code the more common form as v_{high} ($x = 1$) and a less common form as v_{low} ($x = 0$), counting both forms with the same rules. Server and tool names are counted after the normalization in Appendix A.1. For parameter schemas, the counting unit depends on the dimension: for naming, we count records that contain each core parameter name (e.g., `to` vs. `recipient_id`); for structuring and decomposition, we count records with each structural form (separate top-level fields vs. a single container, or separate fields vs. a composite field). Counts of exact parameter sets are not used to order structural forms.

Descriptions. Descriptions are coded by completeness rather than frequency. The v_{low} variant briefly states the tool's function; the v_{high} variant adds explanations of parameters and, for Send Email and Get Weather, return values, following prior work on description quality (Hasan et al., 2026).

Non-target factors. In all single-factor comparisons, non-target factors use their v_{low} variants (Section 4). When P is varied, both candidates keep the brief description, so parameter names are not repeated in the description. When D is varied, both candidates use the v_{low} parameter schema, so the v_{high} descriptions below refer to v_{low} parameter names. In the factorial experiment (Section 5.3), each v_{high} description instead lists the parameters of the schema assigned to that candidate.

Candidate identifiers. A stable two-letter suffix distinguishes each candidate's server name. In server-name comparisons, the suffix is held fixed while the base name changes. The exposed tool identifier joins the suffixed server name and the tool name with `__` (e.g., `message-server-xl__send_gmail`). The suffix does not encode a metadata variant or a presentation position.

B.2 TASK-SPECIFIC VARIANTS

Table 3: Metadata variants for the five tasks. Server names are shown without the candidate-specific suffix. Description entries summarize the content; the full text appears in Appendix B.3.

Task	Factor	v_{high} (1)	v_{low} (0)
Send Email	S	`email-server`	`message-server`
	T	`send_email`	`send_gmail`
	D	Parameter and return explanations	Brief description
	P	`to, subject, body`	`recipient_id, subject, message`
Get Weather	S	`weather-server`	`utility-server`
	T	`get_weather`	`weather_current`
	D	Parameter and return explanations	Brief description
	P	`location`	`input`
Create Issue	S	`github-server`	`pygithub-server`
	T	`create_issue`	`create_github_issue`
	D	Parameter explanations	Brief description
	P	Separate parameters: `owner, repo, title, body`	One dictionary parameter: `params_dict`
Place Order	S	`order-server`	`shop-server`
	T	`place_order`	`create_order`
	D	Parameter explanations	Brief description
	P	Separate parameters: `customer_id, item_id, quantity`	One object parameter: `create_order_requests`
Create Pull Request	S	`github-server`	`github-enterprise`
	T	`create_pull_request`	`create_pr`
	D	Parameter explanations	Brief description
	P	`owner, repo, head, base, body`	`repository, body, base, head`

B.3 Complete Descriptions and Parameter Schemas

The following listings reproduce the description wording, field types, required fields, and property order used in the experiments. As explained in Appendix B.1, the v_{high} descriptions refer to the v_{low} parameter names.

B.3.1 Send Email

The parameter contrast changes `recipient_id` to `to` and `message` to `body`. All three fields are required strings in both variants. The v_{high} description also rewords the first line ("Send an email.").

Description, v_{low}.

```
Send a new email.
```

Description, v_{high}.

```
Send an email.

Args:
    recipient_id: Recipient email address
    subject: Email subject
    message: Email body content

Returns:
    Confirmation of email sent
```

Parameter schema, v_{low}.

```
{
  "type": "object",
  "properties": {
    "recipient_id": {"type": "
    string"},
    "subject": {"type": "string"},
    "message": {"type": "string"}
  },
  "required": [
    "recipient_id",
    "subject",
    "message"
  ]
}
```

Parameter schema, v_{high}.

```
{
  "type": "object",
  "properties": {
    "to": {"type": "string"},
    "subject": {"type": "string"},
    "body": {"type": "string"}
  },
  "required": [
    "to",
    "subject",
    "body"
  ]
}
```

B.3.2 GET WEATHER

The parameter contrast changes the single required string field from `input` to `location`.

Description, v_{low}.

```
Get current weather for a location.
```

Description, v_{high}.

```
Get current weather for a location.
Args:
    input (str): City name
Returns:
    dict: WeatherAPI current weather JSON.
```

Parameter schema, v_{low}.

```
{
  "type": "object",
  "properties": {
    "input": {"type": "string"}
  },
  "required": [
    "input"
  ]
}
```

Parameter schema, v_{high}.

```
{
  "type": "object",
  "properties": {
    "location": {"type": "string"}
  },
  "required": [
    "location"
  ]
}
```

B.3.3 CREATE ISSUE

v_{low} exposes an unrestricted dictionary parameter, `params_dict`; its schema permits additional keys without enumerating the inner fields. v_{high} explicitly declares four required string fields. The description of the dictionary explains the expected issue fields.

Description, v_{low}.

```
Create a new issue within a GitHub repository.
```

Description, v_{high}.

```
Create a new issue within a GitHub repository.

Args:
    params_dict (dict): Issue parameters containing these fields:
        owner (str): Repository owner's user or organization name
        repo (str): Repository name
        title (str): Issue title
        body (str): Issue body text
```

Parameter schema, v_{low}.

```
{
  "type": "object",
  "properties": {
    "params_dict": {
      "type": "object",
      "additionalProperties": true
    }
  },
  "required": [
    "params_dict"
  ]
}
```

Parameter schema, v_{high}.

```
{
  "type": "object",
  "properties": {
    "owner": {"type": "string"},
    "repo": {"type": "string"},
    "title": {"type": "string"},
    "body": {"type": "string"}
  },
  "required": [
    "owner",
    "repo",
    "title",
    "body"
  ]
}
```

B.3.4 PLACE ORDER

Both variants specify the same three required fields and types. v_{low} groups them under `create_order_requests`; v_{high} exposes them directly as top-level properties.

Description, v_{low}.

```
Place a new order
```

Description, v_{high}.

```
Place a new order

Args:
    create_order_requests (object): Order request containing these fields
    :
        customer_id (str): Numeric customer ID
        item_id (str): Numeric item ID
        quantity (int): Quantity of items
```

Parameter schema, v_{low}.

```
{
  "type": "object",
  "properties": {
    "create_order_requests": {
      "type": "object",
      "properties": {
        "customer_id": {"type": "
    string"},
        "item_id": {"type": "string
    "},
        "quantity": {"type": "
    integer"}
      },
      "required": [
        "customer_id",
        "item_id",
        "quantity"
      ]
    }
  },
  "required": [
    "create_order_requests"
  ]
}
```

Parameter schema, v_{high}.

```
{
  "type": "object",
  "properties": {
    "customer_id": {"type": "string
    "},
    "item_id": {"type": "string"},
    "quantity": {"type": "integer"}
  },
  "required": [
    "customer_id",
    "item_id",
    "quantity"
  ]
}
```

B.3.5 CREATE PULL REQUEST

v_{low} represents the repository as one `repository` string in `owner/repo` format. v_{high} uses separate `owner` and `repo` strings.

Description, v_{low}.

```
Create a new pull request
```

Description, v_{high}.

```
Create a new pull request

Args:
    repository (str): Repository identifier in owner/repo format
    body (str): Pull request body text
    base (str): Target branch to merge changes into
    head (str): Source branch containing the proposed changes
```

Parameter schema, v_{low}.

```
{
  "type": "object",
  "properties": {
    "repository": {"type": "string
    "},
    "body": {"type": "string"},
    "base": {"type": "string"},
    "head": {"type": "string"}
  },
  "required": [
    "repository",
    "body",
    "base",
    "head"
  ]
}
```

Parameter schema, v_{high}.

```
{
  "type": "object",
  "properties": {
    "owner": {"type": "string"},
    "repo": {"type": "string"},
    "head": {"type": "string"},
    "base": {"type": "string"},
    "body": {"type": "string"}
  },
  "required": [
    "owner",
    "repo",
    "head",
    "base",
    "body"
  ]
}
```

C ADDITIONAL RESULTS

C.1 THREE-TOOL SELECTION RATES FOR CREATE ISSUE

Figure 7 reports the observed selection rates behind the Create Issue estimates in Table 2, using the definitions in Section 3. Each factor shows a distinct pattern.

Parameter schemas. All five models select a v_{high} tool, which exposes separate top-level parameters, in 93.3%–100% of trials in both compositions. Gemini and Qwen do so in every trial, which is why their P estimates are separated.

Descriptions. v_{high} selection rates are generally above the composition-specific baselines, most clearly for Gemini (93.3% and 84.0%) and Qwen (92.7% and 92.0%).

Tool names. v_{high} selection rates stay near the baselines (33.3%–45.3% for 1H+2L and 63.3%–67.3% for 2H+1L), whereas GPT, Claude, DeepSeek, and Qwen select the first-listed tool in 78.7%–96.0% of trials.

Server names. Claude, DeepSeek, and Qwen again select the first-listed tool in most trials (88.5%–100%). GPT instead favors the v_{high} server name (95.3% and 99.3%), and Gemini the v_{low} one (3.3% and 21.3% v_{high}).

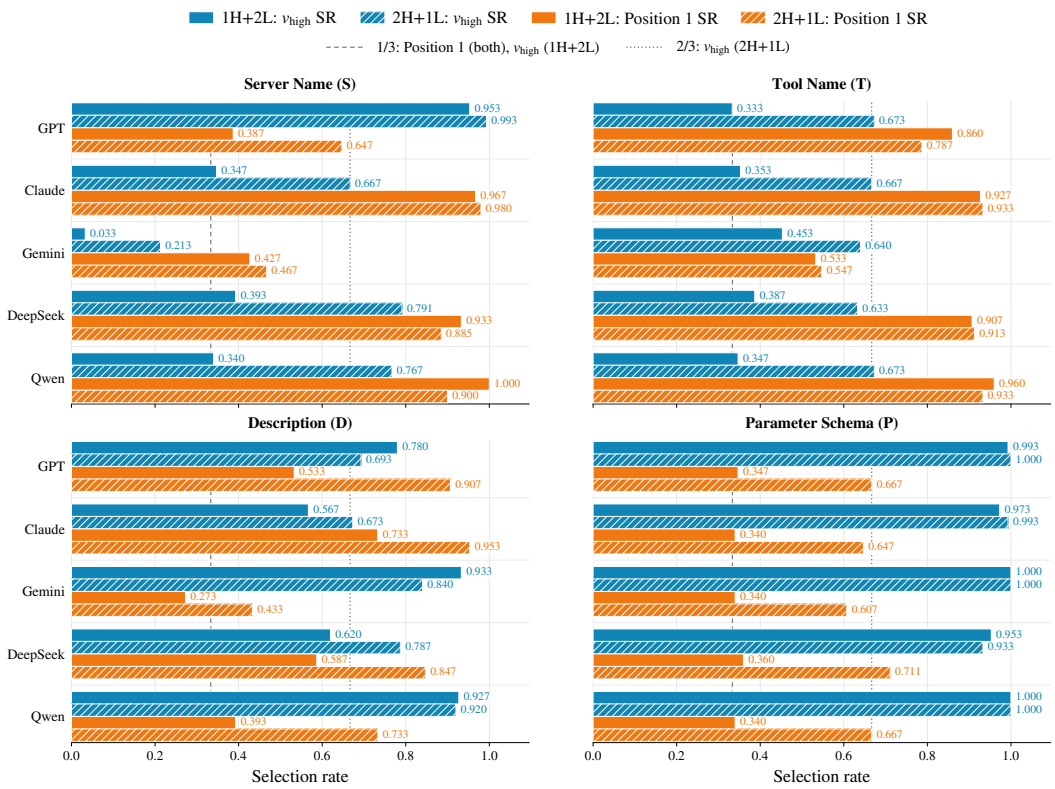


Figure 7: Observed three-tool selection rates for Create Issue across models and metadata factors. Blue bars show the v_{high} selection rate and orange bars the position-1 selection rate; solid and hatched bars denote the 1H+2L and 2H+1L compositions, respectively. Each bar summarizes up to 150 trials. Under uniform choice, the v_{high} baseline is $1/3$ for 1H+2L and $2/3$ for 2H+1L, and the position-1 baseline is $1/3$ for both.

C.2 Send Email Coefficients Across Candidate-Set Sizes

Table 4 reports the Send Email coefficients with two, three, and five candidate tools. The two- and five-tool rows correspond to Figure 2a, and the three-tool rows repeat Table 2.

Parameter schemas. P is positive in all 15 model–size combinations and is the largest estimate in every row except Qwen with five tools. With three and five tools, P excludes zero for GPT, Claude, Gemini, and DeepSeek and is the largest estimate in each of these eight rows. The other P estimates, all two-tool estimates (6.48–11.24) and Qwen's with three and five tools (11.77 and 12.69), are large but have intervals that include zero, reflecting near-separation rather than weak effects. The effect does not change uniformly with the number of candidates: from three to five tools, P increases for GPT and DeepSeek but decreases for Claude and Gemini.

Descriptions and names. D excludes zero for GPT, Gemini, and DeepSeek with three and five tools (0.55–2.49) and for Gemini with two tools (1.59), and it is always smaller than P in these rows. Claude's D never excludes zero. T excludes zero only for GPT at all three set sizes (1.08–1.98) and for Gemini with three and five tools (1.62 and 1.20). S excludes zero only for GPT and Gemini with three and five tools and for Claude with five tools (1.35–2.39). None of DeepSeek's or Qwen's name estimates excludes zero.

Two candidate tools. Only 2 of the 20 two-tool estimates exclude zero: GPT's T (1.08) and Gemini's D (1.59). Claude's and Qwen's S and D estimates are exactly zero, which occurs when choices are fully determined by position, and several others, such as DeepSeek's S (-5.97) and D (5.97), are large with intervals that include zero. With two tools, position therefore masks most metadata effects in Send Email, as in Place Order (Section 5.2).

Table 4: Conditional-logit estimates of metadata effects for Send Email with two, three, and five candidate tools.

Model	Tools	S	T	D	P
GPT-5.4 mini	2	$4.72^{\dagger}$	**1.08**	$0.22^{\dagger}$	$8.15^{\dagger}$
	3	2.39	1.98	1.37	**4.13**
	5	1.79	1.69	0.63	**5.17**
Claude Sonnet 4.6	2	$0.00^{\dagger}$	$4.72^{\dagger}$	$0.00^{\dagger}$	$7.79^{\dagger}$
	3	$1.74^{\dagger}$	$1.24^{\dagger}$	$0.00^{\dagger}$	**4.56**
	5	1.35	$0.07^{\dagger}$	$0.12^{\dagger}$	**3.15**
Gemini 3.5 Flash	2	$5.97^{\dagger}$	$0.19^{\dagger}$	**1.59**	$11.24^{\dagger}$
	3	1.54	1.62	0.55	**7.88**
	5	2.05	1.20	0.70	**6.96**
DeepSeek V4 Pro	2	$-5.97^{\dagger}$	$3.91^{\dagger}$	$5.97^{\dagger}$	$7.74^{\dagger}$
	3	$0.01^{\dagger}$	$-0.83^{\dagger}$	2.49	**3.03**
	5	$0.13^{\dagger}$	$0.18^{\dagger}$	0.59	**3.71**
Qwen3.7-Plus	2	$0.00^{\dagger}$	$-0.36^{\dagger}$	$0.00^{\dagger}$	$6.48^{\dagger}$
	3	$5.64^{\dagger\ddagger}$	$-0.30^{\dagger}$	$5.64^{\dagger\ddagger}$	$11.77^{\dagger}$
	5	$5.95^{\dagger}$	$0.13^{\dagger}$	$14.44^{\dagger}$	$12.69^{\dagger}$

Note. Entries are conditional-logit coefficients (β_f) for v_{high} vs. v_{low}, adjusted for position. S: server name; T: tool name; D: description; P: parameter schema. †95% confidence interval includes zero; large estimates with † reflect near-separation rather than weak effects. ‡As in Table 2, Qwen's choices are determined by position, so these estimates should not be interpreted as effects. Bold marks the largest estimate in each row among those whose interval excludes zero.

C.3 Send Email Coefficients by Candidate Composition

Table 5 splits the three- and five-tool Send Email estimates in Table 4 by candidate composition, that is, by how many candidates use v_{high} (Section 4). Because β_f measures the preference for a v_{high} candidate over a v_{low} one, it can be compared across compositions with different uniform-choice

baselines. We exclude Qwen from the comparisons below: none of its estimates excludes zero, and its three-tool S and D choices are determined by position.

Parameter schemas. Every numerical P estimate is positive. For GPT, Claude, Gemini, and DeepSeek, P excludes zero in 15 of the 16 multi-tool compositions; the exception, Gemini under 1H+2L (14.03), reflects near-separation. P is also the largest of the four estimates in each of these 16 model–composition pairs except DeepSeek under 1H+2L, where D (3.86) and P (3.85) are nearly equal. The schema preference therefore holds whether v_{high} schemas are in the minority or the majority.

Descriptions. Description effects depend on composition. For GPT, Gemini, and DeepSeek, D is positive and excludes zero whenever v_{high} descriptions are in the minority (1H+2L: 2.17–3.86; 2H+3L: 1.15–2.12). When they are in the majority (2H+1L and 3H+2L), these estimates are near zero or negative, and Gemini favors the v_{low} description (-1.83 and -0.55, both excluding zero). A more complete description thus helps a tool when few candidates have one, but not when most do. Pooling over compositions masks this pattern: Gemini's three-tool estimate of 0.55 combines 3.50 and -1.83. Claude's D never excludes zero.

Names. Server-name estimates are larger when most candidates use the v_{high} server name, for GPT and Claude at both set sizes and for Gemini and DeepSeek with five tools (e.g., Claude: 0.00 vs. 2.88 with three tools and 0.45 vs. 2.08 with five). DeepSeek's five-tool S reverses sign, from -0.78 under 2H+3L to 1.01 under 3H+2L, both excluding zero, so its pooled estimate (0.13) does not. Tool-name effects are concentrated in GPT and Gemini: T excludes zero for GPT in every composition (1.08–2.51) and for Gemini in three of the four multi-tool compositions (0.99–3.34), but for DeepSeek only under 3H+2L (0.59) and never for Claude.

Overall, schema effects are the largest and most stable across compositions, whereas description and server-name effects depend on how the variants are distributed among the candidates.

Table 5: Conditional-logit estimates of metadata effects for Send Email by candidate composition.

(a) Server name (S)

Model	2 Tools	3 Tools		5 Tools	
	1H+1L	1H+2L	2H+1L	2H+3L	3H+2L
GPT-5.4 mini	4.72†	1.99	3.02	1.41	2.45
Claude Sonnet 4.6	0.00†	0.00†	2.88	0.45†	2.08
Gemini 3.5 Flash	5.97†	1.53†	1.55†	1.74	2.45
DeepSeek V4 Pro	−5.97†	0.01†	0.00†	−0.78	1.01
Qwen3.7-Plus	0.00†	7.10†‡	14.34†‡	6.26†	12.55†

(b) Tool name (T)

Model	2 Tools	3 Tools		5 Tools	
	1H+1L	1H+2L	2H+1L	2H+3L	3H+2L
GPT-5.4 mini	1.08	2.51	1.21	1.78	1.55
Claude Sonnet 4.6	4.72†	1.21†	1.27†	−0.21†	0.35†
Gemini 3.5 Flash	0.19†	3.34	0.16†	1.36	0.99
DeepSeek V4 Pro	3.91†	0.00†	−1.55†	−0.24†	0.59
Qwen3.7-Plus	−0.36†	0.78†	−1.33†	0.28†	−0.01†

(c) Description (D)

Model	2 Tools	3 Tools		5 Tools	
	1H+1L	1H+2L	2H+1L	2H+3L	3H+2L
GPT-5.4 mini	0.22†	2.17	0.35†	1.15	0.06†
Claude Sonnet 4.6	0.00†	−0.02†	0.01†	0.24†	−0.01†
Gemini 3.5 Flash	1.59	3.50	−1.83	2.12	−0.55
DeepSeek V4 Pro	5.97†	3.86	0.00†	1.43	−0.37†
Qwen3.7-Plus	0.00†	sep‡	sep‡	14.31†	13.72†

(d) Parameter schema (P)

Model	2 Tools	3 Tools		5 Tools	
	1H+1L	1H+2L	2H+1L	2H+3L	3H+2L
GPT-5.4 mini	8.15†	4.04	4.50	5.02	5.60
Claude Sonnet 4.6	7.79†	4.71	4.39	3.04	3.27
Gemini 3.5 Flash	11.24†	14.03†	5.41	7.88	5.76
DeepSeek V4 Pro	7.74†	3.85	1.90	4.05	3.26
Qwen3.7-Plus	6.48†	12.76†	10.35†	sep	sep

Note. Entries are conditional-logit coefficients (β_f) for v_{high} vs. v_{low}, adjusted for position and estimated separately for each composition; kH+lL denotes k candidates with v_{high} and l with v_{low}. The two-tool column repeats Table 4. †95% confidence interval includes zero; large estimates with † reflect near-separation rather than weak effects. sep: separation, so no finite estimate exists. ‡As in Table 2, Qwen's choices are determined by position, so these estimates should not be interpreted as effects.

C.4 Parameter Names and Field Descriptions

Figure 8 tests whether field descriptions change the naming effects in Section 5.4. In Send Email, the base tool uses `to`, `subject`, and `body`, and the other tool renames `to` to `recipient_id`, `body` to `message`, or both. Each comparison is run without field descriptions and with the same

descriptions on corresponding fields of both tools; all other metadata are identical. We use the 50 Send Email queries, with each tool in each position, giving 300 trials per model.

Recipient name. Without field descriptions, all five models select the base tool in 89.8%–99.0% of trials, consistent with Section 5.4. With field descriptions, this preference largely disappears (50.0%–60.0%). The penalty for `recipient_id` thus stems from how models interpret the name: once both fields carry the same description, the name no longer matters.

Body name. Renaming `body` to `message` has little effect. Base-tool rates are 49.0%–51.5% without field descriptions, except 60.0% for Gemini, and 50.0%–51.5% with them. Like `receiver` and `text`, `message` preserves the field's role.

Both names. Renaming both fields yields base-tool rates of 82.8%–99.0% without field descriptions, no higher than for the recipient rename alone, so the effect is driven by `recipient_id`. With field descriptions, the rates fall to 50.0%–62.0%.

Field descriptions therefore remove naming effects that arise from misleading parameter names, in contrast to their inconsistent effect on structured parameters in Place Order (Section 5.4).

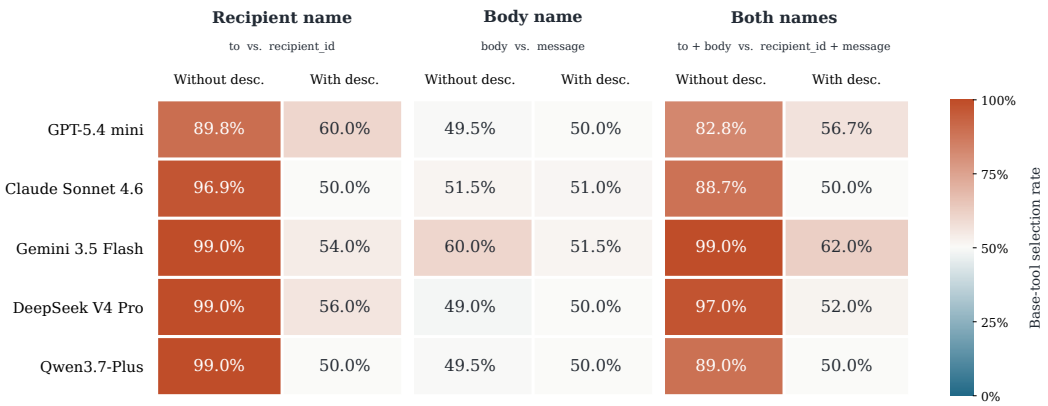


Figure 8: Effects of parameter names and field descriptions on tool selection in Send Email. Cells show the percentage of valid trials in which the model selects the base tool, which uses `to`, `subject`, and `body`. The other tool renames `to` to `recipient_id` (Recipient name), `body` to `message` (Body name), or both (Both names). *With desc.* gives corresponding fields of both tools the same descriptions; *Without desc.* omits field descriptions. The tool-level description is the same in all conditions, and 50% indicates no preference.

C.5 Task Requirements and Parameter Names

Figure 9 combines the task-requirement comparison of Section 5.4 with the `recipient_id` rename. The design follows Appendix A.4, except that the extended tool uses `recipient_id` instead of `to`, so it both supports CC and carries the less preferred recipient name.

CC unspecified. Without a CC request, the extended tool is selected in only 3.0%–25.0% of trials, compared with 45.0%–63.1% when it uses `to` (Section 5.4). The penalty for `recipient_id` thus persists even when the tool offers an additional optional field.

CC required. When the query explicitly requests CC recipients, selection of the extended tool rises for every model. Gemini and Qwen select it in every valid trial, but GPT, Claude, and DeepSeek do so in only 53.0%–67.4% of trials and otherwise choose the base tool, which has no `cc` field. By contrast, all five models select the extended tool in every valid trial when it uses `to` (Section 5.4). A misleading parameter name can therefore lead models to choose a tool that cannot fulfill an explicit request.

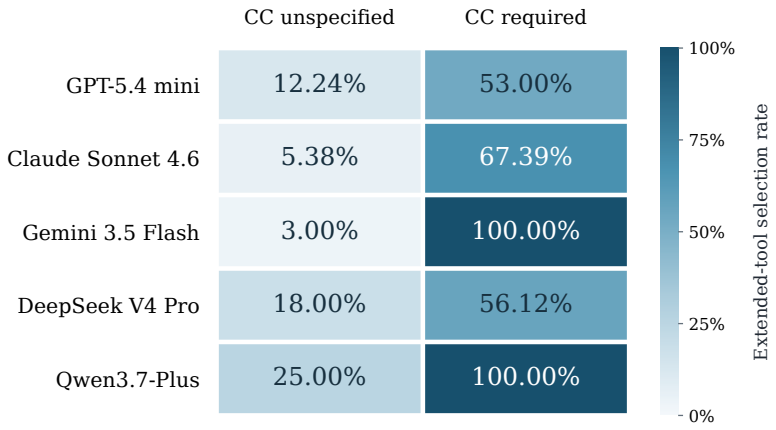


Figure 9: Effects of explicit CC requests on tool selection in Send Email when the extended tool renames `to`. The base tool uses `to`, `subject`, and `body`; the extended tool uses `recipient_id`, `subject`, `body`, and an optional `cc` field. Neither tool has field descriptions. Queries either do not mention CC (CC unspecified) or explicitly request CC recipients (CC required). Cells show the percentage of valid trials in which the model selects the extended tool. Valid trials contain exactly one call to a candidate tool, regardless of argument validity; each condition has 92–100 valid trials out of 100 (e.g., Qwen's 100% under CC required is 97 of 97 valid trials).

D TOOL SELECTION IN MCP

This appendix outlines how an LLM selects tools in MCP (Model Context Protocol, 2026b;a) and which part of this process our experiments examine. Figure 10 illustrates the steps.

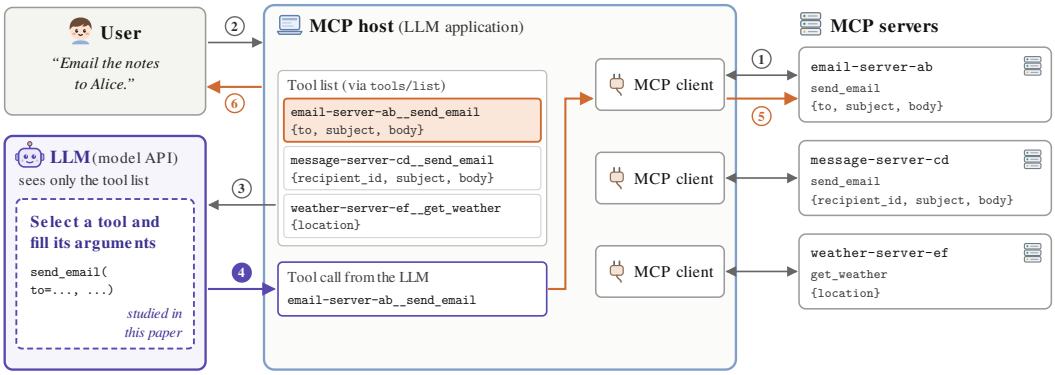


Figure 10: Tool selection in MCP. (1) For each connected server, the MCP host uses an MCP client that calls `tools/list` to obtain each tool's name, description, and input schema. (2) The user sends a request. (3) The host joins server and tool names into a single name field and sends the request with all tools to the LLM. (4) The LLM, which sees only this metadata, selects a tool, fills its arguments, and returns the tool call to the host; this is the step we study. (5) The host invokes the selected tool by sending `tools/call` to its server through the corresponding client, and (6) the result is returned to the user. Descriptions are omitted from the tool list for space.

Components. An MCP *host* is the LLM application with which the user interacts. The host connects to each MCP *server* through its own MCP *client*, and each server exposes one or more tools. The LLM is accessed by the host, typically through a model API, and does not communicate with clients or servers directly.

Discovery and presentation. Before a request is handled, each client calls `tools/list` on its server, which returns every tool's name, description, and input schema (step 1 in Figure 10). When the user sends a request (step 2), the host passes it to the LLM together with the metadata of all available tools (step 3). Because tools from different servers can share a name, the host must also indicate which server each tool belongs to; in our experiments, as in Figure 10, it joins the server and tool names into a single name field (e.g., `email-server-ab__send_email`). The LLM therefore sees exactly the four fields we manipulate: the server name (S) and tool name (T) in the joined name field, the description (D), and the parameter schema (P).

Selection. The LLM chooses one tool and fills its arguments based only on this metadata and the request, and returns the tool call to the host (step 4). It does not observe the tools' implementations, so functionally similar tools can be distinguished only by how their metadata is written. This selection step is the focus of our study.

Invocation. The host executes the tool call by sending `tools/call` with the chosen tool name and arguments to the corresponding server through its client (step 5). The result returns to the host, which delivers the final response to the user, typically after passing the result back to the LLM (step 6).

Scope of our experiments. Our experiments reproduce steps 3 and 4: each request presents the candidate tools' metadata to the LLM, and we record the tool it calls. No tool is executed, so our measurements reflect selection alone, independent of execution outcomes or feedback.