

Agent-Readiness of Synthesized API Tools: Parametric Gravity and the Limits of Documentation Grounding

Teodora Stereciu^{1,2*} Baohao Liao^{1,2†} Christian Herold^{3‡}

Shahram Khadivi² Christof Monz¹

¹Language Technology Lab, University of Amsterdam ²eBay ³Superhuman

Abstract

We study whether LLM agents can autonomously synthesize deployment-ready MCP (Anthropic, 2024) (Model Context Protocol) servers from raw REST API documentation, and whether the resulting servers are reliable enough for downstream agentic task completion. Evaluating 8 synthesis models across 14 APIs under three documentation conditions — full docs, no docs, and counterfactually mutated docs — we find that synthesis is broadly feasible: 11 of 14 APIs achieve sufficient tool coverage, and synthesized servers match or exceed human-authored references. The dominant mediating factor is not model capability but *parametric gravity*: for well-known APIs (GitHub, Stripe, Shopify), training-time knowledge overrides in-context documentation, rendering documentation provision neutral or harmful. A counterfactual mutation study confirms this: high-familiarity APIs show near-zero adherence to renamed parameters, while low-familiarity APIs show near-complete adherence. Logit-lens probing shows the representations of memory and context become increasingly separable in the final transformer layers, consistent with deep feed-forward MLP modules as the site of parametric association storage. Synthesis coverage (SYNTH) correlates strongly with agent task completion ($r = 0.937$), providing a reliable pre-flight quality signal without requiring agent evaluation.

1 Introduction

Large language models (LLMs) are increasingly deployed as agents that plan, reason, and invoke external tools to act in the world (Brown et al., 2020; Wang et al., 2024). The dominant framing of this capability treats tool availability as a precondition: a fixed function library is provided, and the agent

is evaluated on its ability to select and parametrize from within it (Schick et al., 2023; Luo et al., 2025; Liu et al., 2025). What this framing leaves unexamined is the engineering step that precedes tool use: translating API documentation into callable, protocol-compliant interfaces. In practice this is performed by human engineers and does not scale to the present demands of agentic development.

We study whether this step can be automated. Given raw developer documentation for a REST API, can a synthesis agent produce a complete, runnable MCP server? The desired outcome is not individual function stubs, but a deployable tool collection exposing a typed discovery protocol that a separate evaluation agent can invoke at run-time. We evaluate this capability across 14 APIs and 8 models spanning three capability tiers, under three synthesis conditions: full documentation, no documentation, and documentation that deliberately conflicts with training-time API knowledge through counterfactual parameter renaming.

The no-documentation condition yields the central finding of this work. For well-known APIs such as GitHub and Notion, withholding documentation does not degrade — and in several cases improves — downstream agent task performance. Model weights encode accurate implementations of these APIs, and in-context documentation introduces competing signal that models must adjudicate against strong parametric priors. We term this pull of training-time knowledge over in-context evidence *parametric gravity*, and demonstrate that it is not uniform: it tracks API familiarity and complexity, and can be localized to the final transformer layers via logit-lens probing.

The counterfactual condition isolates the mechanism. When 5–7 core parameter names are renamed in the documentation — placing in-context evidence in explicit conflict with training-time representations — high-familiarity APIs exhibit near-zero adherence to the renamed parameters, pro-

*Work done at eBay, as a thesis project.

†Project lead, correspondence to: b.liao@uva.nl.

‡Work done at eBay.

ducing servers faithful to training-time knowledge rather than to the provided documentation. Low-familiarity APIs exhibit near-complete adherence. This binary split is consistent with the training-frequency account of [Mallen et al. \(2023\)](#) and implies that documentation cannot be assumed to reliably ground synthesis when the model has strong parametric priors about the target API.

Our principal findings are as follows. (i) Synthesis is broadly feasible: 11 of 14 APIs achieve synthesis sufficiency of at least 75% in the documentation condition, with a frontier-tier median agent pass rate of 83%, and synthesized servers match or exceed human-authored reference implementations on GitHub and Zulip. (ii) For high-familiarity APIs, parametric memory substitutes for documentation without degradation. (iii) Mutation adherence approaches zero for well-known APIs and approaches unity for unfamiliar ones. (iv) Synthesis coverage is a strong predictor of downstream agent performance ($r = 0.937$, $p < 0.0001$, $n = 209$), providing a reliable pre-flight quality signal assessable without running the evaluation agent. (v) Logit-lens probing finds that the divergence between memory- and context-driven processing becomes most pronounced in the final transformer layers, where divergence is sharp and late, supporting targeted fine-tuning as a feasible intervention.

2 Related Work

Tool use and synthesis. Extensive prior work evaluates LLM tool-use ability against fixed function libraries ([Schick et al., 2023](#); [Qin et al., 2024](#); [Luo et al., 2025](#); [Liu et al., 2025](#); [Maninger et al., 2025](#); [Liao et al., 2026b, 2025, 2026a](#)). The closest works to our synthesis setting are ToolFactory ([Ni et al., 2025b](#)), which generates callable stubs from unstructured documentation, and Doc2Agent ([Ni et al., 2025a](#)), which scales stub generation across a large number of APIs. Neither produces a complete deployable server, evaluates end-to-end agent task completion against live APIs, nor examines the effect of parametric knowledge on synthesis fidelity. The LATM framework ([Cai et al., 2024](#)) establishes the conceptual separation between tool creation and tool use; we extend it to the live API execution setting where generated tools must satisfy strict interface contracts enforced by real services. Our primary failure mode — tool coverage gaps arising at synthesis time — is antecedent to the implementation errors WAPIIBench ([Maninger](#)

[et al., 2025](#)) characterizes (e.g., illegal endpoint URLs, hallucinated parameter names, missing required arguments).

Parametric knowledge and knowledge conflicts. [Mallen et al. \(2023\)](#) demonstrate that the probability of parametric retrieval scales with entity popularity in pre-training data: common entities are retrieved from weights even when context provides contradictory information. [Longpre et al. \(2021\)](#) find that stronger parametric priors increase the probability of ignoring contradictory passages in reading comprehension settings, a pattern our mutation study replicates in the code-generation domain. [Li et al. \(2025\)](#) introduce a training objective designed to suppress parametric interference during context-grounded generation; our mutation design provides a natural diagnostic for evaluating whether such training objectives transfer to the synthesis domain. We extend this line of inquiry to the code-generation domain, where the consequence of parametric override is not an incorrect factual answer but a deployed server that silently ignores in-context API specifications.

Context faithfulness and long-context limitations. [Xie et al. \(2024\)](#) demonstrate that models can be steered by stated premises; subsequent work has identified conditions under which strong parametric priors override contextual updates ([Sun et al., 2026](#)). [Shi et al. \(2023\)](#) show that irrelevant inserted context degrades retrieval fidelity independently of knowledge conflict. [Liu et al. \(2024\)](#) document a positional U-shaped recall curve in long contexts, relevant to the coverage gaps we observe for large APIs with many endpoints.

Mechanistic interpretability. [Meng et al. \(2022\)](#) localize factual recall to mid-layer MLP modules in GPT-style models, providing a candidate mechanistic account of resistance to contextual override for well-known entities. The logit lens ([nostalgebraist, 2020](#)) and tuned lens ([Belrose et al., 2023](#)) enable layer-wise observation of evolving token predictions in the residual stream, forming the methodological basis for our probing study. [Du et al. \(2025\)](#) demonstrate that instruction fine-tuning alters residual stream geometry in ways that modulate parametric association strength, consistent with the widespread in adherence rates we observe across fine-tuned models with comparable pre-training exposure.

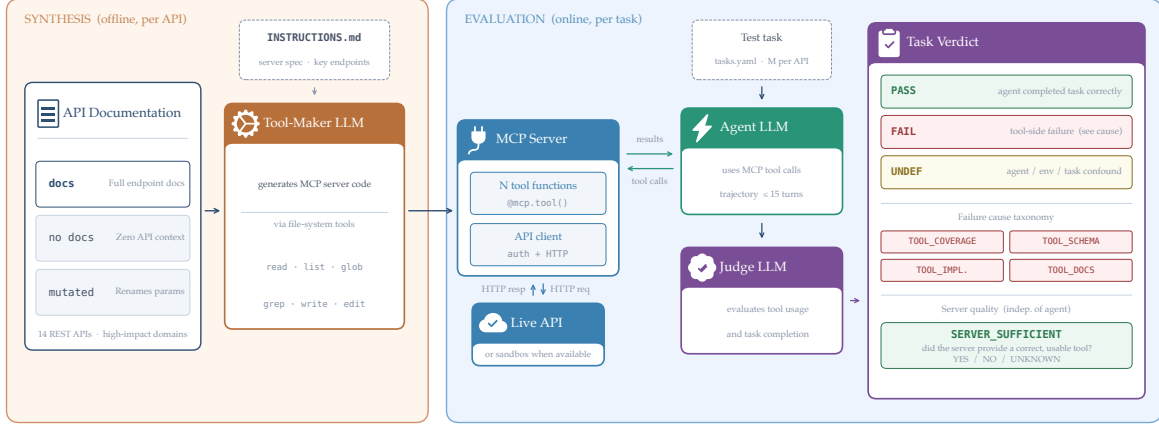


Figure 1: End-to-end benchmark pipeline. A synthesis agent (“Tool-Maker”) reads documentation and produces an MCP server; a separate evaluation agent (“Agent”) attempts tasks against it; a judge LLM scores each attempt on tool usage and task completion independently.

3 Benchmark and Experimental Design

Figure 1 illustrates the end-to-end pipeline. We now describe each component.

APIs and documentation. We evaluate 14 REST APIs¹ spanning five domains: developer tooling (GitHub, Jira), productivity (Notion, Confluence), e-commerce (eBay Buy/Commerce/Sell, Shopify), communication (Mastodon, Zulip), and financial/data services (Stripe, Alphavantage, Spoonacular, OpenWeatherMap). APIs are selected to span two key structural axes: *familiarity* (pre-training data representation) and *complexity* (schema breadth, parameter nestedness, and state/authentication overhead). We operationalize familiarity via an empirical composite density score $S_{comp} \in [0, 1]$ derived from log-normalized GitHub implementation hits (N_{GH}) and Stack Overflow question counts (N_{SO}), partitioning APIs into High, Medium, and Low tiers via quantile split (Table 1). Structural complexity varies naturally across domains, ranging from low-complexity, read-heavy data services (e.g., OpenWeatherMap, Alphavantage) to high-complexity enterprise platforms featuring nested schemas and multi-resource state management (e.g., Stripe, Shopify, Jira). Official developer documentation is scraped to raw Markdown (4–190 files/API) without selective filtering, capturing navigation artifacts and multi-language samples to evaluate realistic synthesis conditions.

¹We use a simplified count here; some items on the list are in fact multi-API context groups, in which case we target all underlying RESTful services (e.g. eBay Buy’s Deal, Feed, Marketing, Order, Offer, Browse APIs).

Table 1: Empirical Familiarity Tier Categorization of Evaluated REST APIs based on Stack Overflow (N_{SO}) and GitHub Code Search (N_{GH}) Density Proxies.

API	N_{SO}	N_{GH}	S_{comp}	Tier
GitHub	3,762	4,784,128	0.914	High
Stripe	12,548	111,360	0.852	High
OpenWeatherMap	838	348,672	0.705	High
Notion	299	38,720	0.545	High
Shopify	753	7,312	0.544	High
eBay Buy	1,104	2,480	0.529	Medium
Alphavantage	232	39,232	0.527	Medium
eBay Sell	1,104	1,912	0.519	Medium
eBay Commerce	1,104	493	0.465	Medium
Jira	1,501	64	0.407	Low
Confluence	269	375	0.354	Low
Spoonacular	20	18,944	0.328	Low
Zulip	10	1,248	0.174	Low
Mastodon	67	14	0.129	Low

Synthesis pipeline. A synthesis agent built on a minimal file-system scaffold (six tools: read, write, edit, glob, grep, list) receives a task specification (Appendix B) and, depending on the condition, a documentation directory. The LLM iterates within this scaffold until it produces three required deliverables: `server.py` (the MCP entry point), `requirements.txt` (pinned dependencies), and `grounding.json` (a manifest mapping each tool to its source documentation file and HTTP endpoint). No step-level guidance or chain-of-thought prompting is imposed. We evaluate eight synthesis models spanning three capability tiers across four model families: GPT-5.2, GPT-5.4, and Sonnet 4.6 (frontier); Gemini 3.1 Pro, Haiku 4.5, and Qwen3-Coder-Next (mid); GPT-5.4-mini and Qwen3-Coder-Next-Base (base).

Synthesis conditions. Each API is synthesized under three conditions corresponding to distinct in-

stantiations of the in-context documentation C : the full documentation corpus (C_{docs}), the no-docs condition ($C_{\emptyset}$), and a counterfactually mutated corpus (C_{mut} , mutated). In $C_{\emptyset}$, raw endpoint documentation and OpenAPI schemas are withheld; the synthesizer receives only high-level API metadata and task specifications (e.g., target domain, base URLs, authentication schemes, and requested broad capabilities), forcing it to rely on parametric memory for schema structure and parameter naming. In C_{mut} , 5–7 core parameter names are renamed throughout the documentation using exact-string substitution. Replacement names are syntactically plausible and consistent with adjacent API conventions, while being unambiguously distinct at the string level (e.g., `per_page` $\rightarrow$ `items_per_page`, `state` $\rightarrow$ `issue_status`). A synthesis model following the documentation should produce tools using the renamed parameters; a model dominated by parametric memory should reproduce the original names regardless.

Evaluation. A separate evaluation agent (Gemini 2.5 Pro) connects to the synthesized server via MCP’s stdio transport and attempts 13–25 natural-language tasks per API (265 tasks total; Appendix D). Tasks are designed to require tool selection without naming tools, and synthesize over retrieved results rather than echo them (Appendix C). A third LLM (Gemini 3 Flash) judges each attempt (Appendix E), returning a structured verdict: PASS/FAIL, a failure cause drawn from seven categories (e.g., `TOOL_COVERAGE`, `TOOL_SCHEMA`, `TASK_AMBIGUOUS`; see Appendix A), and a server sufficiency verdict independent of what the agent did, based only on the list of exposed tools. Judge reliability is validated in Appendix I. We further quantify documentation following in the mutated condition as *mutation adherence* at two levels: interface adherence (renamed identifier in the Python tool signature) and HTTP adherence (renamed string in the outbound HTTP payload). HTTP adherence is always $\leq$ interface adherence; the gap between them indicates partial documentation following.

Metrics. **PASS** is the fraction of task attempts judged PASS out of all scoreable attempts; failures attributed to `AGENT_REASONING`, `ENVIRONMENT`, or `TASK_AMBIGUOUS` are excluded from both numerator and denominator, isolating synthesis infrastructure quality. **SYNTH** (synthesis sufficiency) is the fraction of tasks for which the judge found the

server sufficient to complete the task, independent of agent behavior; it measures synthesis quality without running the evaluation agent.

Human reference baselines. For GitHub and Zulip, publicly available community-maintained MCP servers are evaluated against the same task suite and judge pipeline without modification. These baselines represent hand-crafted implementations produced with API familiarity and full documentation access, providing an upper-bound estimate for what manual synthesis can achieve.

4 Behavioral Results

4.1 Synthesis Feasibility

Synthesis is broadly feasible for the majority of APIs tested. Eleven of 14 APIs achieve $\text{SYNTH} \geq 75\%$ in the documentation condition under the frontier tier, with a frontier median PASS of 83%. The dominant failure mode is *coverage gap* — the server starts and executes correctly, but endpoints required by certain tasks are absent — rather than structural failure (non-executable servers). Coverage gaps are most severe for Jira (SYNTH 56%, PASS 50%) and eBay Sell (SYNTH 43%, PASS 38%), both of which have large hierarchical documentation corpora exceeding what the synthesizer can attend to in a single pass, consistent with a negative log-linear relationship between endpoint count and PASS. Across the remaining 12 APIs, structural synthesis failures are rare for frontier models; they become more frequent at lower tiers, where base-tier models sometimes terminate without producing a runnable `server.py`.

Figure 2 shows median PASS per synthesis model under both conditions. Frontier models cluster at 83–97% in the docs condition; the no-docs condition meets or exceeds docs for all three frontier models (GPT-5.2: 88 vs. 83%; GPT-5.4: 90 vs. 74%; Sonnet 4.6: 87 vs. 97%). Among mid-tier models, Qwen3-Coder-Next reaches near-frontier performance (80%) even without documentation, while Haiku 4.5 drops from 62% to 40%, suggesting a partially formed but insufficient parametric representation. Base-tier models collapse without documentation (Qwen3-Coder-Next-Base: 45% $\rightarrow$ 0%), confirming that strong parametric priors are a prerequisite for documentation-free synthesis.

Synthesized servers match or exceed the two available human-authored references. The official

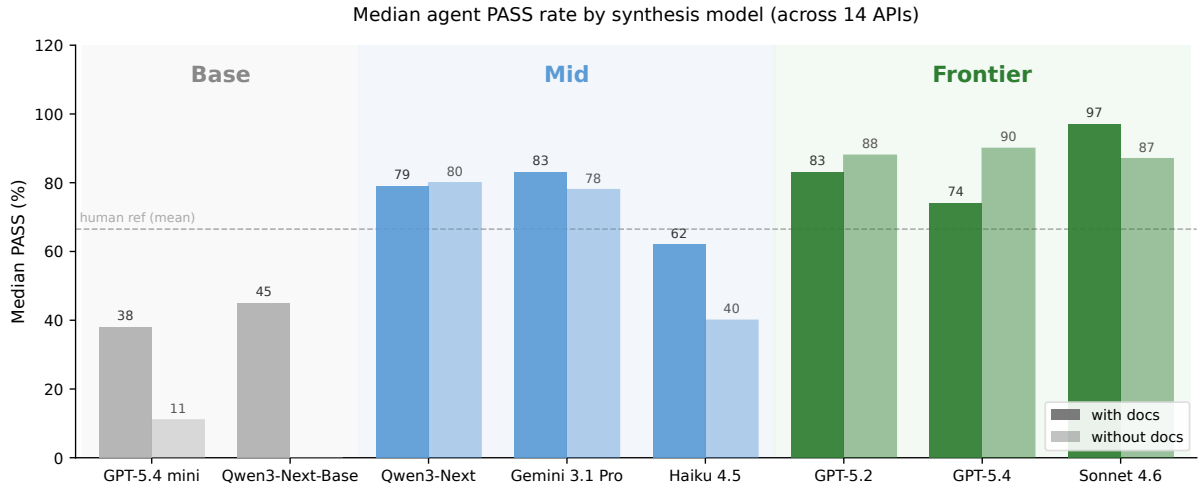


Figure 2: Median agent PASS rate per synthesis model under docs (solid) and no-docs (faded) conditions, grouped by model tier. Frontier models achieve docs-condition medians of 83–97% and maintain exceptional performance even without documentation (87–90% PASS), with GPT-family models matching or exceeding their doc-guided baseline. Across tiers, doc-free synthesis scales sharply with model capacity; base-tier models collapse without documentation, suggesting that strong parametric memory is a prerequisite for documentation-free tool generation.

GitHub MCP server achieves 65% PASS; synthesized servers without documentation comfortably exceed this (81–90% across repeated runs), because parametric knowledge of the GitHub API produces more complete endpoint coverage than the human-authored reference. The community Zulip server achieves 68% PASS; the synthesized documentation-condition server scores 80%. In both cases, the human reference fails on the same pattern: it covers the core API surface but omits less common task-relevant endpoints (webhooks and Actions dispatch for GitHub; message editing and DM search for Zulip). Synthesis stochasticity beyond the main single-run results is characterized in Appendix H. Individual generations exhibit non-trivial output variance across conditions, including occasional startup degeneracy or complete tool failure. We primarily attribute this sampling noise to an underdeveloped synthesis harness, which lacks automated verification or self-correction loops.

4.2 Parametric Gravity: Adherence and Performance Consequences

The counterfactual condition reveals a sharp, familiarity-stratified split in documentation adherence (Figure 3). High-familiarity APIs — such as GitHub, Stripe, and Shopify — cluster below 20% adherence across all synthesis models; lower-familiarity APIs cluster predominantly above 30%. The gradient broadly follows pre-training exposure (S_{comp}): core high-familiarity APIs strongly resist

documentation edits, while adherence increases markedly for lower-familiarity endpoints before plateauing across Medium (58%) and Low (56%) tiers.

Model-level variation within this structure is substantial, as Figure 3 shows. GPT-5.4 is the most documentation-faithful synthesis model overall, reaching 100% adherence for Confluence, Mastodon, and eBay Commerce. Gemini 3.1 Pro is the clear outlier: near-zero adherence across all APIs and familiarity tiers, indicating synthesis predominantly from parametric memory regardless of API exposure. Across the high-familiarity tier, the mean adherence is 27% (with primary web APIs like Shopify, GitHub, and Stripe remaining below 20%): parametric gravity at this tier is not a GPT-family artifact but a property of API familiarity that persists across model families and capability tiers.

The counterfactual condition reveals a sharp, familiarity-stratified split in documentation adherence (Figure 3). High-familiarity APIs — GitHub, Stripe, Shopify — cluster below 20% adherence across all synthesis models; low-familiarity APIs cluster above 70%. The gradient is near-monotone: no low-familiarity API falls below 40% and no high-familiarity API exceeds 20%.

Within this structure, model-level adherence varies across targets (Appendix G), ranging from GPT-5.4 achieving 100% adherence on select APIs to Gemini 3.1 Pro exhibiting near-zero adherence across all conditions.

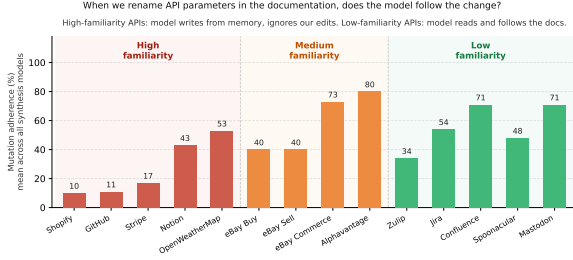


Figure 3: Mean interface-layer adherence (%) to mutated parameter names per API, ordered by pre-training familiarity (S_{comp}). High-familiarity APIs (e.g., Shopify, GitHub, Stripe) strongly resist documentation edits ($< 20\%$ adherence) due to strong parametric priors. Lower-familiarity APIs adhere significantly more ($> 30\%$). Adherence plateaus between Medium (58%) and Low (56%) tiers. Tier means: High 27%, Medium 58%, Low 56%.

The performance consequences of low adherence are non-monotonic. GitHub exhibits near-zero adherence and is *unaffected* by mutation (+14 pp under GPT-5.2): both the synthesizer and the evaluation agent operate from the same accurate training-time representation, rendering the mutation effectively invisible to the full pipeline. Stripe also exhibits near-zero adherence yet still *loses* 15 pp under mutation. The mechanism is not parameter-name mismatch but synthesis-level noise: contradictory signals between parametric priors and mutated documentation degrade other aspects of server quality — endpoint path construction, authentication handling — without appearing in the adherence metric. High adherence is neither necessary nor sufficient for immunity to mutation-induced degradation.

4.3 Documentation Availability

The aggregate effect of including documentation is smaller than expected. Frontier-tier condition medians are: docs 83%, no-docs 88%, mutated 77%. High-familiarity APIs drive the no-docs advantage — GitHub and Notion improve when documentation is withheld (by 21 and 22 pp under GPT-5.2), because documentation removal reduces the competing context that parametric memory must override. The reverse holds for lower-familiarity APIs: Jira and Zulip lose 20–25 pp under no-docs, indicating genuine dependence on in-context documentation that parametric memory cannot compensate for. While broadly applicable, there are some exceptions to this effect, due to structural complexity. Stripe, for instance, is a counterexample: although

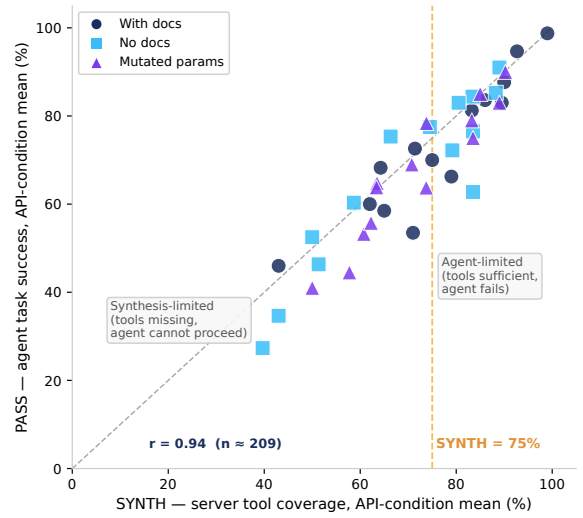


Figure 4: SYNTH vs. PASS. Plotted points are (API, condition) means across frontier and mid-tier models; Pearson $r = 0.937$ is computed from all 209 raw (model, API, condition) triples. The diagonal marks $y = x$; the vertical dashed line at SYNTH = 75% separates synthesis-limited cases (left) from agent-limited cases (right of line, below diagonal). Marker shape encodes synthesis condition. Note: Part of the correlation is structurally driven by tool coverage failures directly lowering PASS outcomes.

highly familiar, its structural complexity still demands endpoint documentation. Below SYNTH = 75%, low-familiarity APIs consistently require documentation; its removal degrades performance in all cases.

4.4 Synthesis Quality as a Predictor of Agent Performance

SYNTH and PASS correlate strongly across all 209 (model, API, condition) triples ($r = 0.937$, $p < 0.0001$). Figure 4 shows this relationship. Points fall tightly along the diagonal, with two interpretable off-diagonal regions: the lower-left quadrant (low SYNTH, low PASS) is labeled “Synthesis-limited”; coverage gaps in the server propagate directly to agent failure. Points to the right of SYNTH = 75% that fall below the diagonal (high SYNTH, lower PASS) are labeled “Agent-limited”; the server is sufficient, but the agent fails to exploit it; these tend to be frontier models on complex multi-step tasks.

The vertical dashed line at SYNTH = 75% functions as a reliable threshold: entries to its left correspond to PASS below 60% in nearly all cases, identifying synthesis-limited cases before committing to full agent evaluation.

5 Mechanistic Analysis

The behavioral split in Section 4 — near-zero adherence for GitHub and Shopify versus near-complete adherence for Confluence and eBay Commerce — raises a mechanistic question: at which point in the transformer forward pass do parametric representations separate from in-context evidence? Answering this matters for two reasons: it confirms whether the behavioral split is representationally grounded or an artifact of synthesis noise, and it identifies the layer range where a targeted fine-tuning intervention would be feasible. We address this through a logit-lens probing study (nostalgebraist, 2020; Belrose et al., 2023) on Qwen3-Coder-Next, one of the eight synthesis models evaluated in the main benchmark, enabling us to trace the representational origin of the behavioral patterns we observe.

Setup. For each mutated synthesis context, we identify probe positions where original and mutated parameter names are in direct lexical competition, specifically the token position immediately preceding the target name. To prevent trivial logit equivalence, we exclude mutation pairs that share an identical first BPE token. At each layer ℓ , we apply the final layer normalization to the residual state $\mathbf{h}_\ell$ before projecting through the unembedding matrix W_U (pre-LN logit-lens) to compute

$$\delta_\ell = \text{logit}_\ell(\text{original}) - \text{logit}_\ell(\text{mutated}),$$

where a positive value indicates the parametric (training-time) token is preferred and a negative value indicates the contextual (documentation-specified) token is preferred.

We probe $N = 38$ valid token positions across 11 APIs on Qwen3-Coder-Next (deployed locally via vLLM on $2 \times \text{H100}$ GPUs, using standard greedy decoding $T = 0$ and a 512-token sequence window), which is tasked with the generation of one select MCP tool (per-API; Appendix J). Stripe and Mastodon are excluded due to synthesis failure, and Shopify due to tokenization overlap between all mutation pairs.

API-level memory fractions. The left panel of Figure 5 shows the fraction of probed mutations where the model’s final-layer representation favors the original training-time parameter name over the mutated documentation name. The split closely mirrors the behavioral adherence results:

GitHub and eBay Commerce exhibit 100% parametric memory across all probed mutations, while AlphaVantage, Jira, OpenWeatherMap, eBay Sell, and Spoonacular show 100% context-following. Mixed APIs (Confluence, eBay Buy, Zulip: 20–25% memory; Notion: 50%) exhibit parameter-level variation, consistent with individual parameter familiarity rather than a uniform API-level disposition. This confirms that the adherence gap observed behaviorally is not an artifact of synthesis quality but is rooted in how the model internally represents API parameter names.

Late-layer commitment. The right panel of Figure 5 traces δ_ℓ across all 49 layers for four representative mutations. The central finding is that δ_ℓ remains low-magnitude and near-zero throughout the first $\sim 80\%$ of the network in every case, regardless of the eventual outcome. Divergence between parametric and contextual representations emerges primarily in the final 6–10 layers (layers 40–48), consistent with late-layer factual association mechanisms (Meng et al., 2022).

The two GitHub mutations (red) illustrate clean parametric capture: δ_ℓ is near-zero through layers 0–38 before rising sharply and monotonically into the decision zone, reaching large positive values by layer 48. The lon→longitude trace (green solid) shows the complementary context-following pattern: δ_ℓ becomes consistently negative from around layer 6 and drops sharply in layers 41–48 as the documentation token longitude is fully committed.

The units→measurement_units trace (green dashed) illustrates an important intermediate case. The probe is taken at the input position immediately preceding measurement_units in the documentation, so the logit lens measures the model’s internal representation *while reading the docs*, not during generation. The trace goes positive in the decision zone ($\delta_{46} \approx +9$), indicating that parametric memory for units activates at the doc-reading position. Yet the generation outcome is context-following: the model writes measurement_units in the synthesized function signature. This dissociation shows that a positive δ at the probe position is necessary but not sufficient for a memory outcome: the generation is determined by the accumulated context across the full input, which for a simple API like OpenWeatherMap is strong enough to override the parametric attractor that manifests at the probe position.

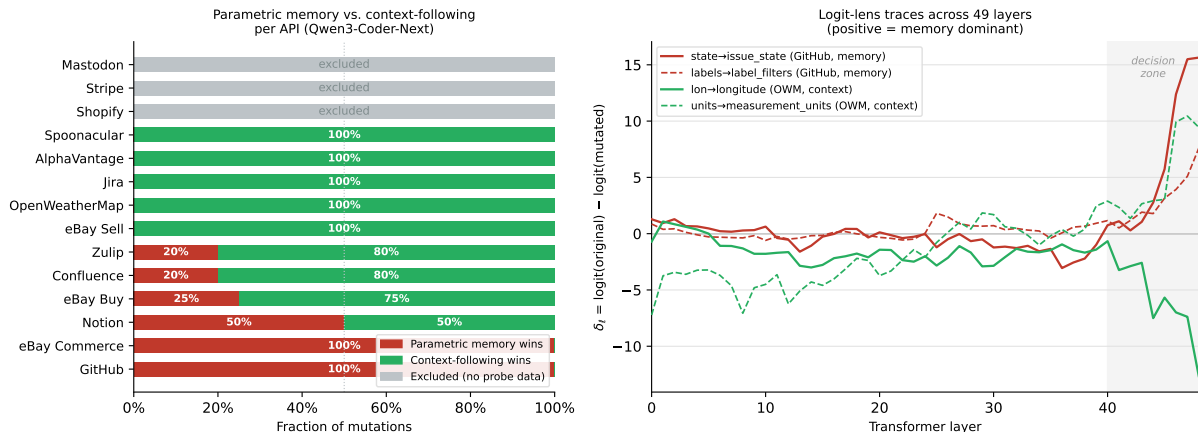


Figure 5: *Left*: Fraction of probed mutations where parametric memory overrides the mutated documentation, per API (Qwen3-Coder-Next, 49 layers, 11 APIs). Red = memory wins; green = context-following wins; grey = excluded (no valid probe positions). The split mirrors the behavioral adherence results from Figure 3. *Right*: Logit-lens traces $\delta_\ell = \text{logit}_\ell(\text{original}) - \text{logit}_\ell(\text{mutated})$ across all 49 layers for four representative mutations. Positive δ = parametric memory dominant; negative δ = context-following. All traces are near-zero through layers 0–38; the outcomes strongly diverge in the shaded decision zone (layers 40–48, ≈ 82 –98% of network depth). The green dashed trace (units→measurement_units) goes positive in the decision zone yet the generation outcome is context-following, illustrating that parametric activation at the probe position does not guarantee a memory outcome when the accumulated contextual signal is strong enough to override it at generation time.

6 Discussion and Implications

Across 14 APIs and 8 synthesis models, agent task success is governed primarily by API familiarity — the completeness of the model’s pre-training representations — and structural complexity, rather than raw model capability. High-familiarity APIs achieve strong performance even without documentation, while low-familiarity or structurally complex APIs depend on explicit guidance; the counterfactual mutation study further shows that documentation cannot be assumed to override parametric priors when the two conflict. The SYNTH-PASS correlation ($r = 0.937$) and the late-layer separation pattern together provide both a practical quality signal and a mechanistic account of why the familiarity gap exists.

Implications for synthesis system design. For high-familiarity APIs, documentation adds noise rather than signal: the GitHub no-docs condition (81–90% PASS) comfortably exceeds both the docs condition and the human-authored reference (65%), because documentation introduces competing signal without adding coverage. For less popular or actively versioned APIs, documentation provides grounding that parametric memory cannot supply. A SYNTH score below 75% reliably identifies synthesis-limited cases before committing to full agent evaluation.

The mechanism of parametric gravity. The late-layer divergence pattern is consistent with factual recall accounts in which parametric associations are encoded as key-value memories in deep feed-forward MLP modules (Meng et al., 2022; Liao et al., 2023; Nichani et al., 2025). Under this account, the residual stream accumulates both parametric and contextual signal through the mid-network, but the final-layer MLP modules (layers 40–48 in Qwen3-Coder-Next) tip the accumulated representation toward the stored parameter schema for well-known APIs, overriding the contextual evidence integrated earlier. The dissociation observed in the units→measurement_units trace — where δ_ℓ goes positive in the decision zone yet the generation is context-following — shows that parametric activation is necessary but not sufficient for a memory outcome: accumulated contextual signal can override a localized parametric attractor, but only when that signal is strong enough to compete.

Implications for model choice. The tier structure in Figure 2 shows a clear capability gap: base-tier models consistently underperform the 66.5% human reference mean in the docs condition, while all mid-tier and frontier models exceed it. Within frontier models, the no-docs condition is surprisingly competitive — GPT-5.2 scores 88% without documentation versus 83% with it — consistent

with parametric gravity substituting for documentation when training-time API knowledge is accurate. The widespread in adherence rates across synthesis models with comparable capability suggests that post-training alignment substantially modulates context faithfulness independently of raw PASS performance, pointing to fine-tuning in the final “commitment” zone as a targeted intervention.

7 Conclusion

We have presented an end-to-end benchmark for automated MCP server synthesis and characterized the primary factor mediating synthesis reliability: the accuracy and completeness of the synthesis model’s training-time representation of the target API. Synthesis is broadly feasible for the majority of REST APIs evaluated — 11 of 14 APIs achieve sufficient tool coverage, and frontier and mid-tier synthesized servers match or exceed human-authored references (GitHub: 65%; Zulip: 68%) — but the decision of whether to include documentation is not neutral. For high-familiarity APIs, parametric memory constitutes an accurate and sufficient substitute; for less familiar and complex APIs, documentation provides grounding that weights cannot supply. The SYNTH-PASS correlation ($r = 0.937$) establishes synthesis coverage as a reliable pre-flight quality signal, and logit-lens probing connects the behavioral adherence split to late-layer decision dynamics in the transformer forward pass.

As MCP adoption accelerates, the automated synthesis capability we evaluate becomes increasingly relevant to production deployment, which also surfaces governance questions our benchmark does not address: how synthesized server correctness is verified at scale, how API versioning is tracked, and re-synthesis is triggered, whether synthesis-at-scale creates systemic attack surfaces through shared parametric errors, and who bears accountability when a synthesized tool silently calls the wrong endpoint. These questions sit at the intersection of AI systems and software engineering governance and represent an important direction for the field as MCP transitions from research prototype to infrastructure.

Limitations

The benchmark covers 14 APIs selected to vary in familiarity and complexity, but does not include APIs with non-standard authentication schemes

(OAuth 2.0 device flows, mutual TLS), streaming endpoints, or non-REST contracts. Evaluation runs against live or sandbox API endpoints, introducing flakiness from rate limits, transient errors, and account-state dependencies. Since evaluations rely on single-run synthesis, results should be interpreted with appropriate caution given the non-trivial sampling variance and startup degeneracy introduced by model stochasticity and harness interactions. The benchmark evaluates technical synthesis quality, leaving out deployment-level properties such as security, versioning correctness, or access-control compliance; governance of synthesized MCP servers in production is an open problem. Judge reliability is assessed through a judge-sensitivity study (varying the judge model), but judge scores remain an imperfect proxy for real-world task success. Finally, the probing study uses Qwen3-Coder-Next (49 layers), a model in the mid tier of the synthesis benchmark; the mechanistic findings may not generalize to frontier-scale closed models whose internal architecture differs substantially. Logit-lens interpretability assumes approximate linearity of the residual stream projection, which may not hold uniformly across layers or model scales.

References

- Anthropic. 2024. Model context protocol. <https://modelcontextprotocol.io>.
- Nora Belrose, Zach Furman, Logan Smith, Danny Halawi, et al. 2023. Eliciting latent predictions from transformers with the tuned lens. In *Workshop on Challenges in Deployable Generative AI (ICML)*.
- Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, et al. 2020. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901.
- Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024. Large language models as tool makers. In *The Twelfth International Conference on Learning Representations*.
- Hongzhe Du, Weikai Li, Min Cai, Karim Saraipour, Zimin Zhang, Himabindu Lakkaraju, Yizhou Sun, and Shichang Zhang. 2025. How post-training reshapes LLMs: A mechanistic view on knowledge, truthfulness, refusal, and confidence. In *Conference on Language Modeling*.
- Gaotang Li, Yuzhong Chen, and Hanghang Tong. 2025. Taming knowledge conflicts in language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*. Spotlight.

- Baohao Liao, Hanze Dong, Christof Monz, Xinxing Xu, Li Dong, and Furu Wei. 2026a. Multi-turn on-policy distillation with prefix replay. *arXiv preprint arXiv:2607.04763*.
- Baohao Liao, Hanze Dong, Xinxing Xu, Christof Monz, and Jiang Bian. 2026b. [Self-hinting language models enhance reinforcement learning](#). *CoRR*, abs/2602.03143.
- Baohao Liao, Yan Meng, and Christof Monz. 2023. [Parameter-efficient fine-tuning without introducing new latency](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, ACL 2023, Toronto, Canada, July 9-14, 2023, pages 4242–4260. Association for Computational Linguistics.
- Baohao Liao, Yuhui Xu, Hanze Dong, Junnan Li, Christof Monz, Silvio Savarese, Doyen Sahoo, and Caiming Xiong. 2025. [Reward-guided speculative decoding for efficient LLM reasoning](#). In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net.
- Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.
- Wenrui Liu, Zixiang Liu, Elsie Dai, Wenhan Yu, Lei Yu, Tong Yang, Jinjun Han, and Hong Gao. 2025. MCPAgentBench: A benchmark for evaluating LLM agents in the MCP ecosystem. *arXiv preprint arXiv:2512.24565*.
- Shayne Longpre, Kartik Perisetla, Anthony Chen, Nikhil Ramesh, Chris DuBois, and Sameer Singh. 2021. Entity-based knowledge conflicts in question answering. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 7052–7063.
- Ziyang Luo, Zhiqi Shen, Wenzhuo Yang, Zirui Zhao, Prathyusha Jwalapuram, Amrita Saha, Doyen Sahoo, Silvio Savarese, Caiming Xiong, and Junnan Li. 2025. MCP-Universe: Benchmarking agents on real-world MCP servers. *arXiv preprint arXiv:2508.14704*.
- Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When not to trust language models: Investigating effectiveness of parametric and non-parametric memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics*, pages 9802–9822.
- Daniel Maninger, Leon Chemnitz, Amir Molzani Shari-floo, Jannis Brugger, and Mira Mezini. 2025. Benchmarking web API integration code generation. In *Proceedings of the 2nd IEEE/ACM International Conference on AI-Powered Software (AIware)*.
- Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems*, volume 35, pages 17359–17372.
- Xinyi Ni, Haonan Jian, Qiuyang Wang, Vedan-shi Chetan Shah, and Pengyu Hong. 2025a. Doc2Agent: Scalable generation of agents from API documentation. *arXiv preprint arXiv:2506.19998*.
- Xinyi Ni, Qiuyang Wang, Yukun Zhang, and Pengyu Hong. 2025b. ToolFactory: Automating tool generation by leveraging LLM to understand REST API documentation. *arXiv preprint arXiv:2501.16945*.
- Eshaan Nichani, Jason D. Lee, and Alberto Bietti. 2025. Understanding factual recall in transformers via associative memories. In *The Thirteenth International Conference on Learning Representations*.
- nostalgebraist. 2020. Interpreting GPT: The logit lens. *LessWrong*.
- Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.
- Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36.
- Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Schärli, and Denny Zhou. 2023. Large language models can be easily distracted by irrelevant context. In *Proceedings of the 40th International Conference on Machine Learning, ICML*.
- Kaiser Sun, Fan Bai, and Mark Dredze. 2026. Task matters: Context-dependent knowledge conflict resolution in LLMs. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*.
- Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. 2024. Executable code actions elicit better LLM agents. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 50208–50232.
- Jian Xie, Kai Zhang, Jiangjie Chen, Renze Lou, and Yu Su. 2024. Adaptive chameleon or stubborn sloth: Revealing the behavior of large language models in knowledge conflicts. In *The Twelfth International Conference on Learning Representations*.

A Failure Cause Taxonomy

Each task attempt is assigned exactly one verdict by the judge LLM: `PASS` or one of seven failure causes. The taxonomy is defined at two levels: *synthesis failures* (attributable to the synthesized server) and *evaluation exclusions* (excluded from both numerator and denominator of `PASS`).

Synthesis failures (counted in `PASS` denominator):

- **TOOL_COVERAGE:** The server does not expose a tool capable of performing the required action. The task cannot proceed regardless of agent behavior.
- **TOOL_SCHEMA:** A relevant tool exists, but its parameter names, types, or required fields are incorrect, preventing the agent from constructing a valid call. Includes cases where mutated parameter names are absent from the tool signature.
- **TOOL_IMPLEMENTATION:** The tool schema is correct, but the underlying implementation fails — wrong HTTP method, incorrect endpoint path, malformed authentication header, or missing request body serialization.
- **TOOL_OUTPUT:** The tool executes and returns a response, but the output format or field names do not match what the agent or downstream task expects (e.g., paginated results where a flat list is required).

Evaluation exclusions (excluded from `PASS` numerator and denominator):

- **AGENT_REASONING:** The server is sufficient — the judge independently verifies that a correct tool call sequence exists — but the agent fails to construct or execute it. Includes tool selection errors, incorrect parameter values, and failure to chain multi-step results. Excluded because the failure is not attributable to synthesis quality.
- **ENVIRONMENT:** The failure is attributable to an external condition rather than synthesis or agent quality: API rate limits, network timeouts, sandbox permission errors, or transient service unavailability.
- **TASK_AMBIGUOUS:** The task specification is insufficiently constrained to admit a unique correct answer, or requires information not available in the API surface (e.g., account-specific data not present in the test environment).

The `SERVER_SUFFICIENT` verdict is recorded independently of the task outcome and forms the basis of the `SYNTH` metric: the judge evaluates the tool list against the task specification without running the agent, returning `SUFFICIENT` or `INSUFFICIENT` with a justification.

B Synthesis Task Specification

Each synthesis run receives an `INSTRUCTIONS.md` file (one per API). All APIs share the same template; the example below is for Zulip, which also serves as the evaluation task dataset in Appendix C. In the no-docs condition, the agent is also instructed with the following: “NOTE: No API documentation is available for this task – the docs/ directory is empty. Ignore any references to documentation files in the task below. Use your general knowledge of the API to implement as many tools as you can.\n\n”

Task: Build an MCP Server for Zulip REST API

What You're Building

An MCP server with comprehensive coverage of the Zulip REST API, suitable for use by an autonomous agent completing real-world tasks.

Choose your implementation language and framework:

- Python: FastMCP (from `mcp.server.fastmcp` import `FastMCP`)
- TypeScript: official MCP SDK (`@modelcontextprotocol/sdk`)

The server must run over stdio and expose tools callable via the MCP protocol.

What You Have

- 165 API endpoint documentation files in docs/
- Environment variables for authentication:
 - `ZULIP_EMAIL` -- bot or user email address
 - `ZULIP_API_KEY` -- API key
 - `ZULIP_SITE` -- Zulip instance URL
- Base URL: `{ZULIP_SITE}/api/v1`

Authentication

Zulip uses HTTP Basic Auth with email and API key. See the docs for details.

Coverage Expectations

- Aim for broad coverage of the most important operations, not minimal coverage
- Prioritize: create, read, update, delete on core resources
- Include tools useful for multi-step workflows
- Cover: Messages (send, edit, delete, move, fetch history, flags), Streams (create, subscribe, archive, settings), Topics (list, rename, resolve, delete), Users (profiles, presence), Reactions, Scheduled Messages, File Uploads

Technical Requirements

- Discoverability: all tools accessible via `list_tools()`
- Return format: JSON-serializable results
- Error handling: return errors as dicts (e.g. `{"error": "..."}`) -- do not raise unhandled exceptions
- No generic passthrough tools: every exposed tool must correspond to a specific, named operation

Deliverables

Always produce:

- `grounding.json` -- maps every tool to its source doc and HTTP endpoint:

```
{
  "send_message": {
    "doc": "docs/messages/send-message.md",
    "endpoint": "POST /messages"
  }
}
```
- `server.py` -- MCP server entry point (stdio)
- `requirements.txt` -- pinned dependencies:

```
fastmcp==3.2.4
requests==2.32.3
```

C Evaluation Task Examples

Each API has 13–25 evaluation tasks. Below is a representative subset for Zulip (20 tasks total), covering all tag categories used across the benchmark. Each task prompt is given verbatim to the evaluation agent; the `expected` field is used only by the judge to verify the response.

READ

- id: `list_streams`
prompt: >
List all streams (channels) available in this Zulip organization. Report the names of the first 5 streams you find.
expected: `general`
tags: `[read]`
- id: `get_topics_in_stream`
prompt: >
Retrieve all topics that have been used in the "general" stream. Report the topic names.
expected: `name`
tags: `[read, multi-step]`

WRITE

- id: `send_stream_message`
prompt: >
Send a message to the stream "general" with topic "agent-test" and content "Hello from the MCP agent harness! (automated test)". Report the message ID of the sent message.
expected: `message_id`
tags: `[write]`
- id: `add_alert_word`
prompt: >
Add "agent-eval-keyword" as an alert word for

the current user. Then retrieve the current list of alert words to confirm it was added. Report the full list.

expected: `agent-eval-keyword`
tags: `[write, multi-step, workflow]`

PAGINATION

- id: `paginate_stream_messages`
prompt: >
Fetch messages from the "general" stream in two batches using anchor-based pagination: first request the oldest 10 messages (anchor "oldest", `num_after=10`), then use the ID of the last message as the anchor for a second request. Report the message ID of the 10th and 11th messages.
expected: `message_id`
tags: `[read, pagination, edge-case]`

ERROR HANDLING

- id: `send_to_nonexistent_stream`
prompt: >
Attempt to send a message to a stream called "zzz-this-stream-does-not-exist-xyz999". Do not retry. Report exactly what the server returns, including any error code or message.
expected: `error`
tags: `[error-handling, edge-case]`

HARDER: CROSS-ENDPOINT LINKAGE

- id: `message_edit_history`
prompt: >
Send a message to "general" with topic "mcp-eval-history-test" and content "Original version". Edit it twice: first to "Second version", then to "Third version". Retrieve the edit history and report how many edits appear and the `prev_content` of the most recent edit entry.
expected: `Second version`
tags: `[write, multi-step, workflow, harder]`
- id: `move_message_topic`
prompt: >
Send a message to "general" with topic "mcp-eval-move-source" and content "This message will be moved". Move it to topic "mcp-eval-move-dest" in the same stream by editing only its topic. Verify the move by fetching messages narrowed to the new topic. Report the message ID and the new topic name.
expected: `mcp-eval-move-dest`
tags: `[write, multi-step, workflow, harder]`

D Evaluation Agent Prompt

For each evaluated MCP server, the agent attempting the tasks sees the following prompt.

You are an autonomous agent completing tasks by calling API tools. You **MUST** use the \ provided tools -- do not answer from prior knowledge or describe what you would do. \ Call tools to get real data and take real actions.

Rules:

1. Always call at least one tool before giving a final answer.
2. Use results from earlier tool calls to inform later ones.
3. When a task contains an ALL_CAPS name like GITHUB_TEST_REPO, ADYEN_MERCHANT_ACCOUNT, \ZULIP_EMAIL, etc. -- pass that exact string as the argument value. Do NOT ask what it \ means, do NOT wait for clarification. Just pass "GITHUB_TEST_REPO" (or whatever the \ name is) directly to the tool. The server handles resolution.
4. When done, summarize what the tools returned.

Never say "I cannot" or "I would need". Never ask for clarification. Just call the tools.

E Judge Prompt

The agent attempt as well as server sufficiency is judged by a judge following the prompt below.

You are an objective judge evaluating an AI agent's task completion.
Follow the output format exactly. Do not add any extra text or markdown.

Task given to agent:
{task.prompt}{expected_hint}

Tools available in this MCP server:
{tool_list}

Agent trajectory:
{trajectory}

Answer the following three questions.

1. OUTCOME: Did the agent complete the task correctly?
PASS if the final answer correctly addresses the task based on real tool results.
FAIL if the task was not completed, completed incorrectly, or the agent could not use the tools.
2. FAILURE_CAUSE: If FAIL, what caused it? Choose the single best fit:
TOOL_COVERAGE - no tool exists in the server for the required operation
TOOL_DOCUMENTATION - a relevant tool exists but poor naming, descriptions, or undocumented parameters caused the agent to miss it , call the wrong one, or pass wrong values.
Use this when a capable agent would have succeeded with clearer tool design.
TOOL_SCHEMA - tool schema has structural problems: wrong param types, missing required fields, or a response format the agent cannot parse
TOOL_IMPLEMENTATION - tool ran but returned wrong, incomplete, or malformed data; includes runtime

crashes and exceptions inside the tool
AGENT_REASONING - server was adequate; agent hallucinated values or ignored a correct tool result.
Reserve for unambiguous hallucinations only.
If the agent was confused by poor tool design, use TOOL_DOCUMENTATION instead. Assume the agent is competent.
TASK_AMBIGUOUS - the task prompt itself was unclear or underspecified [measurement confound]
ENVIRONMENT - server crash on startup, network failure, or missing credentials [measurement confound]
NONE - use if OUTCOME is PASS

3. SERVER_SUFFICIENT: Setting aside what the agent did, did this server have everything needed to complete the task if used correctly?
YES - a tool exists, its schema is correct, it returns valid data, and its name/description are clear enough for a competent agent to find and use it
NO - any of the above conditions fail (missing tool, broken schema, bad implementation, or so poorly described that no competent agent could use it)
UNKNOWN - ENVIRONMENT or TASK_AMBIGUOUS prevented a fair assessment

Respond in exactly this format (no extra text, no markdown):

OUTCOME: PASS or FAIL

FAILURE_CAUSE: <code from list above, or NONE if PASS>

SERVER_SUFFICIENT: YES, NO, or UNKNOWN

REASON: <one sentence explaining the outcome>

F Full Results

Table 2 shows the full per-synthesizer breakdown of agent PASS (%) on the evaluation suite. The human-authored reference servers were also evaluated with the following results: GitHub official server 65% PASS; Zulip community server 68% PASS. The performance gap between the human baseline and the top-performing synthesized servers is mostly explained by tool coverage, showing that indeed integration may be better scaled this way.

Table 3 reports server synthesis sufficiency (SYNTH, %) for every synthesis model, API, and condition. SYNTH is assessed by the LLM judge on the synthesized server's tool list without running the evaluation agent; it measures what fraction of task-relevant tools are present and usable. The

Table 2: Agent PASS (%) per API and synthesis model, organized by tier. In the GPT-5.2 columns, D(S) shows PASS with SYNTH in parentheses for the docs condition. Rows sorted by frontier-tier docs median descending. **Bold** = row maximum PASS. **Red** = below 50%. — = synthesis crash. *Med.* = per-column median across evaluated APIs.

API	Base						Mid						Frontier											
	GPT-5.4-mini			Qwen3-Coder-Next-Base			Qwen3-Coder-Next			Gem 3.1 Pro			Haiku 4.5			GPT-5.2			GPT-5.4			Sonnet 4.6		
	D	N	M	D	N	M	D	N	M	D	N	M	D	N	M	D(S)	N	M	D	N	M	D	N	M
OpenWeatherMap	100	100	100	—	—	—	100	86	100	85	90	95	95	65	60	100 (100)	86	100	100	95	100	—	100	—
eBay Commerce	0	0	0	45	—	—	100	—	11	83	44	45	30	10	50	100 (100)	89	55	88	40	62	100	—	—
Mastodon	57	79	73	0	—	0	87	80	54	100	92	87	93	40	86	100 (93)	100	91	90	100	—	—	92	
Spoonacular	94	82	100	—	60	—	—	—	—	83	91	90	93	79	86	100 (75)	90	78	80	82	60	93	—	92
Zulip	40	14	29	—	—	0	—	—	64	76	84	62	55	36	26	80 (84)	60	62	90	85	71	100	43	—
Shopify	62	0	0	—	0	0	94	82	67	80	70	93	93	75	94	87 (89)	92	62	83	92	53	100	—	—
Alphavantage	—	—	—	92	—	—	—	—	—	100	100	90	—	—	—	86 (84)	90	88	80	92	78	—	—	—
Confluence	33	80	43	64	—	—	39	—	50	100	100	80	33	33	92 (95)	93	91	69	100	85	—	94	47	—
Notion	60	—	—	—	0	—	69	5	—	94	94	100	56	5	50	78 (91)	100	100	62	100	—	100	87	85
Stripe	14	0	19	—	56	—	—	62	—	47	50	59	57	75	86	72 (77)	62	57	8	62	12	77	100	100
eBay Buy	0	0	82	6	—	—	12	80	6	90	67	—	80	67	100	73 (88)	67	80	69	92	100	—	79	—
GitHub	38	32	30	—	—	—	58	60	57	78	71	36	95	90	86	62 (61)	83	76	58	89	73	90	100	100
Jira	0	0	25	—	—	—	71	—	42	58	44	31	62	7	64	50 (56)	25	50	57	50	57	73	58	79
eBay Sell	7	8	8	—	0	—	100	—	—	38	8	23	17	9	36	38 (43)	50	43	29	45	44	—	73	—
Med.	38	11	30	45	0	0	79	80	54	83	78	87	62	40	64	83 (86)	88	77	74	90	71	97	87	92

D = docs, N = no-docs, M = mutated. S = SYNTH.

docs-condition SYNTH for GPT-5.2 is also shown inline in Table 2 in the main text.

“—” indicates synthesis crashed before any tools could be evaluated. A value of 0 indicates the server started but the judge found no task-relevant tools. Models that produced no valid servers in any condition (GPT-4.1, Gemini 2.5 Pro, Gemini 3.5 Flash) are omitted.

G Per-Model Mutation Adherence Results

Table 4 shows the per-API mutation adherence for each synthesis model. Adherence varies widely across targets, ranging from near-zero compliance on high-familiarity APIs like Shopify (10.0%) and GitHub (11.4%) to over 70% on lower-familiarity APIs like Alphavantage (80.0%). Model behaviors also diverge within individual APIs, but established platforms show parametric resistance regardless of model family.

H Synthesis Variance Study

To quantify synthesis stochasticity, we ran GPT-5.2 five times on each of three APIs and all three conditions, holding all other variables fixed. Tables 5–8 report per-run PASS and SYNTH; Table 10 summarizes key statistics.

Shopify. The no-docs condition is the most stable (Table 6): SYNTH locks at 94% on every run ($\sigma = 0$), and PASS ranges 80–94% (mean 90%, $\sigma \approx 6$ pp). Docs condition is high-variance (Table 5): PASS 59–93% ($\sigma \approx 13$ pp), with no degeneracy but large differences between small (131-line,

93%) and large (538-line, 75%) servers. Mutated (Table 7) has one startup crash and PASS 67–92% across non-crash runs; mutation adherence is perfectly stable (0% HTTP / 20% iface on every run).

GitHub. Docs synthesis is highly degenerate (Table 8): 4 of 6 runs produce stub servers with 1–5 tools instead of the 46–61 tools in valid runs. Valid runs score 62–67%, tight with the human MCP reference (65%). No-docs degeneracy rate drops to 0/3; all runs score 81–90%, comfortably exceeding the human reference. Mutated is universally degenerate (7/7 runs, 0–6 tools); the mutation context consistently destabilizes GitHub synthesis for this model.

Stripe. Docs and no-docs each have 1/5 degeneracy (startup crash). Valid docs runs score 46–67%; the bottleneck is AGENT_REASONING (8–14 failures per run), not tool coverage. No-docs valid runs score 38–62%, with lower SYNTH (57–73%) reflecting weaker parametric knowledge of Stripe’s API surface. Mutated shows the widest variance (20–69% across 4 non-degenerate runs); adherence itself varies run-to-run (0–20% HTTP), in contrast to Shopify’s perfectly stable adherence. Table 9 summarizes all three conditions.

I Judge Sensitivity Study

To assess whether results depend on the choice of judge model, we re-ran evaluation of five APIs under all three conditions using four different judge LLMs while holding the synthesis server and agent run fixed. Table 11 reports per-judge PASS; Ta-

Table 3: Server synthesis sufficiency (SYNTH, %) per synthesis model, API, and condition. Red = below 50%. — = synthesis crash. D = docs, N = no-docs, M = mutated. Qwen3-NB = Qwen3-Coder-Next-Base; Qwen3-N = Qwen3-Coder-Next.

API	GPT-5.4-mini			Qwen3-NB			Qwen3-N			Gem 3.1 Pro			GPT-5.2			GPT-5.4			Haiku 4.5			Sonnet 4.6		
	D	N	M	D	N	M	D	N	M	D	N	M	D	N	M	D	N	M	D	N	M	D	N	M
GitHub	35	26	35	—	—	—	52	52	57	74	74	30	61	87	61	57	91	65	96	87	86	91	100	100
Stripe	14	—	23	—	68	—	—	73	—	59	59	68	77	68	73	48	73	27	73	82	95	86	100	100
Zulip	48	24	40	—	—	0	—	—	80	80	88	64	84	48	64	88	88	76	61	40	29	100	48 [†]	—
Alphavantage	—	—	—	95	—	—	—	—	—	94	100	89	84	89	89	95	89	89	—	—	—	95	89	84
Confluence	17	95	69	81	—	—	43	21	38	—	100	100	95	95	95	76	89	89	86	15	32	—	95	50
eBay Buy	10	0	88	6	—	—	7	76	6	86	76	0	88	94	88	71	88	100	94	76	94	—	76	29
eBay Commerce	0	8	0	46	—	—	92	—	31	89	44	54	100	85	54	92	38	77	46	31	69	100	—	—
eBay Sell	7	14	14	—	0	—	71	—	—	14	14	29	43	50	43	29	50	57	29	29	50	—	50	—
Jira	6	0	11	—	—	—	69	—	—	56	62	31	44	56	44	56	62	62	67	13	69	69	44	69
Mastodon	60	80	73	0	—	0	—	73	60	93	93	87	93	100	100	92	92	93	93	33	87	—	—	87
Notion	73	0	11	—	0	—	82	0	—	95	95	100	91	100	100	61	100	—	82	0	67	100	91	91
OpenWeatherMap	100	87	100	—	—	—	100	78	100	83	91	96	100	83	100	100	96	100	96	65	61	—	96	—
Shopify	67	0	0	—	0	0	—	83	72	83	78	94	89	94	61	87	93	56	94	83	94	94	—	—
Spoonacular	94	88	41	—	29	—	—	—	—	88	94	88	75	75	75	88	88	88	0	82	76	94	89	94

Gemini 3.1 Pro Confluence docs: server produced 0 registered tools (import error); SYNTH undefined. Sonnet 4.6 and Haiku 4.5 Alphavantage: 25 req/day API key exhausted; SYNTH from synthesis code inspection. Qwen3-Next-Base and Qwen3-Next: frequent startup crashes reduce coverage; PASS results in Table 2.

Table 4: Interface-layer adherence (%) to mutated parameter names per API across synthesis models with per-API mean adherence across evaluated models.

API	GPT-5.4-mini	Qwen3-Next	Gem 3.1 Pro	Haiku 4.5	GPT-5.2	GPT-5.4	Sonnet 4.6	Mean
eBay Commerce	100	100	0	100	40	100	—	73.3
Shopify	0	0	0	20	20	20	—	10.0
Spoonacular	25	—	14	75	75	75	25	48.2
Zulip	17	100	0	50	17	17	—	33.5
Stripe	20	60	0	20	20	0	0	17.1
Confluence	60	100	0	60	80	100	100	71.4
Jira	80	100	0	80	40	60	20	54.3
Mastodon	100	100	0	0	100	100	100	71.4
OpenWeatherMap	60	40	0	80	80	60	—	53.3
Alphavantage	—	100	0	100	100	80	100	80.0
eBay Sell	40	20	0	40	60	80	—	40.0
eBay Buy	0	80	0	20	60	60	60	40.0
GitHub	0	60	0	20	0	0	0	11.4
Notion	40	0	0	80	80	—	60	43.3

Table 5: Shopify docs variance (5 runs, GPT-5.2). Lines = server.py line count.

Run	Lines	PASS	SYNTH
1	131	93%	94%
2	143	93%	94%
3	522	79%	83%
4	538	75%	83%
5	112	59%	61%
Range		59–93%	61–94%
		$\sigma(\text{PASS}) \approx 13$ pp	

Table 6: Shopify no-docs variance (5 runs, GPT-5.2).

Run	Lines	PASS	SYNTH
1	186	92%	94%
2	230	94%	94%
3	109	93%	94%
4	115	80%	94%
5	151	93%	94%
Range		80–94%	94%
		$\sigma(\text{SYNTH}) = 0$	

Table 7: Shopify mutated variance (5 runs, GPT-5.2, patched). HTTP Adh / Iface Adh stable across all runs.

Run	Lines	PASS	SYNTH	HTTP / Iface
1	142	67%	72%	0% / 20%
2	121	0% (crash)	—	0% / 20%
3	172	92%	94%	0% / 20%
4	174	80%	83%	0% / 20%
5	134	79%	83%	0% / 20%
Range (excl. crash)		67–92%	72–94%	$\sigma(\text{adh}) = 0$

Table 8: GitHub docs variance (6 runs) and no-docs variance (3 runs). Human reference shown for comparison.

Cond	Run	Tools	PASS	SYNTH	Status
—	Human ref	—	65%	65%	—
docs	1	46	67%	70%	—
docs	2	2	—	—	Degenerate
docs	3	3	—	—	Degenerate
docs	4	5	0%	—	Degenerate
docs	5	61	64%	57%	—
docs	6	1	—	—	Degenerate
nodocs	1	63	86%	87%	—
nodocs	2	61	90%	91%	—
nodocs	3	63	81%	83%	—

Table 9: Stripe variance summary across all three conditions (5 runs each, GPT-5.2).

Condition	PASS range	SYNTH range	Degeneracy	Bottleneck
docs	46–67%	68–77%	1/5 (crash)	AGENT_REASONING
nodoccs	38–62%	57–73%	1/5 (crash)	TOOL_COVERAGE
mutated	20–69%	45–77%	1/5 (crash)	Mixed

Table 10: Variance summary statistics across all APIs and conditions. No-docs is the most reliable condition; docs and mutated show higher variance and degeneracy.

Condition	PASS σ (mean)	Degeneracy rate	Primary failure mode
Shopify docs	≈ 13 pp	0/5	Server size variance
Shopify nodocs	≈ 6 pp	0/5	Stable (SYNTH fixed)
Shopify mutated	≈ 11 pp	1/5	Crash, adherence stable
GitHub docs	high	4/6	Degenerate stubs
GitHub nodocs	≈ 4 pp	0/3	Stable, beats reference
GitHub mutated	—	7/7	Universal degeneracy
Stripe docs	≈ 9 pp	1/5	AGENT_REASONING
Stripe nodocs	≈ 9 pp	1/5	TOOL_COVERAGE
Stripe mutated	≈ 17 pp	1/5	Adherence variance

ble 12 summarizes key patterns.

Gemini 3 Flash and GPT-4.1 agree within ± 12 pp on the majority of conditions. Gemini 2.5 Flash is also broadly consistent, with one outlier (Confluence docs: 69% vs. 92–94% for the other three judges) attributable to stricter handling of partial-output responses.

GPT-5.4-mini is a systematic outlier: it produces 7–13 AGENT_REASONING undefined verdicts per run (compared to 0–5 for the other judges), artificially deflating its PASS scores by classifying correct agent answers as incomplete. Manual review of six disputed GitHub docs tasks confirmed that Gemini 3 Flash verdicts were correct in all six cases — GPT-5.4-mini classified clearly present information (issue numbers, PR counts, CI run SHAs) as missing.

Summary. Gemini 3 Flash is a reliable judge. Its scores are consistent with GPT-4.1 and Gemini 2.5 Flash within normal sampling variance. GPT-5.4-mini is not suitable as a judge for this task due to reading-comprehension failures that produce systematically inflated AGENT_REASONING undefined counts.

J Logit-lens Results

Table 13 shows the logit-lens probing results on Qwen3-Coder-Next across APIs and mutated parameter names. Flip layer refers to the lowest layer index where the model’s top prediction flips to the target token.

Table 11: Judge sensitivity study: PASS (%) per judge model across 5 APIs and 3 conditions. AR = AGENT_REASONING undefined count (high AR = judge over-classifies agent answers as incomplete). Baseline judge is Gemini 3 Flash.

API	Cond	Gem 3 Flash	GPT-4.1	GPT-5.4-mini	Gem 2.5 Flash
GitHub	docs	62% (AR=2)	72% (AR=5)	40% (AR=13)	63% (AR=4)
GitHub	nodocs	83% (AR=0)	90% (AR=3)	82% (AR=12)	79% (AR=4)
GitHub	mutated	19% (AR=2)	14% (AR=2)	6% (AR=7)	27% (AR=1)
Stripe	docs	72% (AR=6)	82% (AR=5)	71% (AR=8)	71% (AR=5)
Stripe	nodocs	62% (AR=3)	60% (AR=7)	77% (AR=9)	63% (AR=3)
Stripe	mutated	57% (AR=4)	75% (AR=6)	77% (AR=9)	64% (AR=8)
Zulip	docs	80% (AR=3)	74% (AR=2)	76% (AR=7)	75% (AR=5)
Zulip	nodocs	60% (AR=0)	48% (AR=2)	43% (AR=2)	50% (AR=5)
Zulip	mutated	62% (AR=3)	62% (AR=4)	90% (AR=13)	61% (AR=2)
Shopify	docs	87% (AR=1)	80% (AR=3)	78% (AR=9)	80% (AR=2)
Shopify	nodocs	92% (AR=1)	85% (AR=5)	62% (AR=10)	77% (AR=4)
Shopify	mutated	62% (AR=2)	69% (AR=2)	62% (AR=5)	54% (AR=4)
Confluence	docs	92% (AR=5)	94% (AR=5)	90% (AR=11)	69% (AR=5)
Confluence	nodocs	93% (AR=5)	86% (AR=7)	67% (AR=12)	83% (AR=13)
Confluence	mutated	91% (AR=4)	79% (AR=2)	64% (AR=10)	91% (AR=9)

Table 12: Judge agreement summary. Mean absolute deviation from the Gemini 3 Flash baseline across 15 conditions. GPT-5.4-mini deviates most due to systematic AGENT_REASONING over-classification.

Judge	Mean $ \Delta $ from baseline	Notes
GPT-4.1	8 pp	Broadly consistent; slight positive bias
Gemini 2.5 Flash	7 pp	Consistent; Confluence docs outlier (-23 pp)
GPT-5.4-mini	14 pp	Systematic under-scoring; high AR counts

Table 13: Summary of API Mutation and Probing Results on Qwen3-Coder-Next (49 layers).

Dataset	Function	Mutation	Outcome	Flip Layer	Clean	Skipped
ebay_buy	search_item_summaries	limit → max_results	doc	23	5	0
ebay_buy	search_item_summaries	offset → skip	doc	1	5	0
ebay_buy	search_item_summaries	sort → order_by	doc	40	5	0
ebay_buy	search_item_summaries	filter → search_filter	memory	1	5	0
ebay_buy	search_item_summaries	item_id → listing_id	N/A	N/A	5	0
spoonacular	search_recipes_by_ingredients	query → search_query	N/A	N/A	3	1
spoonacular	search_recipes_by_ingredients	includeIngredients → ingredients	doc	1	3	1
spoonacular	search_recipes_by_ingredients	targetCalories → calorie_target	N/A	N/A	3	1
github	list_repository_issues	per_page → page_limit	memory	3	4	1
github	list_repository_issues	labels → label_filters	memory	4	4	1
github	list_repository_issues	state → issue_state	memory	11	4	1
github	list_repository_issues	sha → commit_sha	N/A	N/A	4	1
notion	query_data_source	filter → query_filter	doc	29	5	0
notion	query_data_source	sorts → sort_rules	doc	4	5	0
notion	query_data_source	page_size → results_per_page	memory	46	5	0
notion	query_data_source	start_cursor → page_cursor	memory	15	5	0
notion	query_data_source	archived → is_archived	N/A	N/A	5	0
zulip	get_messages	narrow → filter_spec	doc	45	6	0
zulip	get_messages	num_before → before_count	doc	41	6	0
zulip	get_messages	num_after → after_count	doc	44	6	0
zulip	get_messages	anchor → start_message_id	doc	13	6	0
zulip	get_messages	topic → message_topic	memory	3	6	0
zulip	get_messages	stream → channel_name	N/A	N/A	6	0
ebay_sell	getFulfillmentPolicies	marketplace_id → market_id	doc	5	3	2
ebay_sell	getFulfillmentPolicies	compliance_type → violation_type	N/A	N/A	3	2
ebay_sell	getFulfillmentPolicies	sku → seller_sku	N/A	N/A	3	2
confluence	search_confluence_content	cql → query	doc	4	5	0
confluence	search_confluence_content	start → offset	doc	1	5	0
confluence	search_confluence_content	limit → max_results	doc	22	5	0
confluence	search_confluence_content	expand → include	doc	40	5	0
confluence	search_confluence_content	status → content_status	memory	1	5	0
alphavantage	time_series_daily_adjusted	symbol → ticker	doc	23	4	1
alphavantage	time_series_daily_adjusted	interval → time_interval	N/A	N/A	4	1
alphavantage	time_series_daily_adjusted	datatype → format	doc	41	4	1
alphavantage	time_series_daily_adjusted	series_type → price_type	N/A	N/A	4	1
openweathermap	get_current_weather	lat → latitude	doc	1	5	0
openweathermap	get_current_weather	lon → longitude	doc	1	5	0
openweathermap	get_current_weather	units → measurement_units	doc	26	5	0
openweathermap	get_current_weather	lang → language	doc	9	5	0
openweathermap	get_current_weather	cnt → count	N/A	N/A	5	0
jira	search_issues_jql	maxResults → page_size	doc	42	3	2
jira	search_issues_jql	fields → include_fields	doc	44	3	2
jira	search_issues_jql	accountId → user_id	N/A	N/A	3	2
ebay_commerce	search_product_summaries	q → query	memory	1	3	2
ebay_commerce	search_product_summaries	aspect_filter → aspects	memory	26	3	2
ebay_commerce	search_product_summaries	limit → page_size	memory	26	3	2