



Tool Search as a Separate Decision Layer

A two-stage evaluation of model-agnostic discovery in a marketing agent

Vikram Sivashankar

Conversion Labs · September 28, 2026

Abstract

As agents gain integrations, their tool catalogs consume an increasing share of context. We study a separate selection step that evaluates a user's request against tool descriptions and loads only the relevant definitions. Our implementation uses Jev, a structured decision model, independently of the model handling the conversation. We compare six methods over a catalog of 114 tools from Conversion: Jev, local lexical and BM25 search, and native OpenAI, Anthropic BM25, and Anthropic regex search. An initial 100-case development suite is followed by 100 new cases authored after freezing the selector and sealed before evaluation. On the fresh suite, Jev achieves complete retrieval on 89/100 cases with 80.4% macro precision, compared with 81/100 and 65.2% for OpenAI native search. On the 95 requests with valid selections from every method, Jev has the highest observed complete-retrieval rate and precision. Its selected definitions average 2.68 kB across the full fresh suite, versus 153.32 kB for the full catalog. At the recorded Jev rate, selection costs approximately \$0.00157 per search. A conversation cost model incorporating cache writes and reads projects a 49.7% to 73.1% reduction in tool-input and search costs for a specified 30-turn scenario. The contribution is a working discovery architecture and a reproducible product-domain evaluation of selection quality, context exposure, and cost.

1 Introduction

Conversion's agents work across contacts, campaigns, workflows, files, and connected services. Every integration adds capabilities: an Airtable integration contributes six tools, and email themes contribute another six. Our agent's catalog reached 114 tools. Most individual requests need only a small subset, yet eagerly supplying the full catalog puts every definition in the caller's context.

This creates a recurring decision: which capabilities should the agent have available for the current request? Keyword matching is inexpensive, but a relevant tool need not share the user's wording. A request to inspect a named Airtable base can require a general `list` tool to discover the base identifier before reading its schema. Conversely, several tools may mention Airtable while operating on the wrong entity.

We investigate whether a small, structured evaluator can make this decision at a cost that supports repeated use throughout a conversation. The selector is separate from the conversation model, allowing either component to change independently. We evaluate capability coverage, irrelevant exposure, unsupported requests, and the cost of selection relative to the tool context it can avoid.

2 Selection and loading

The harness first determines the tools a session is eligible to use. The selector receives that catalog and a search query; the loader exposes selected definitions to the calling model. Tool execution continues through the existing permission and approval checks.

Jev receives shared JSON state containing the query and every tool's name and full description. Each tool has a Boolean question asking whether it directly performs the requested operation or is an indispensable prerequisite. The prompt distinguishes the entity, integration, direction, and read/write intent. It excludes optional investigation and redundant lookup when an identifier is already supplied. Multiple relevant tools or no relevant tool are valid outcomes.

We retain relevance scores of at least 0.5, sort them by descending score with a deterministic name tie-break, and expose at most five tools. All 114 questions fit in one evaluation request. Parameter schemas are held by the loader and supplied only for selected tools; they are not sent to Jev.

The selector contract accepts a catalog and query and returns names and scores. It does not require a particular conversation model. For compatible Anthropic calls, the implementation uses deferred definitions and tool references. Other paths use a client-side set of active tools. Discovery state retains loaded tools across turns and reconstructs them when a conversation resumes, subject to the current eligible catalog.

3 Benchmark design

We use a fixed catalog taken from Conversion's actual tool definitions. Its names, descriptions, and parameter schemas occupy 153,320 bytes when serialized canonically. The evaluation has two stages with the same catalog:

The development suite informed prompt and description changes. We then froze the selector, catalog, threshold, and search budgets before authoring the fresh suite. The fresh requests and labels were sealed before running any method. No selection settings or labels were changed after seeing fresh outputs. The fresh suite covers 92 distinct tools in its required labels and places more weight on multi-

Suite	Single capability	Multiple capabilities	No suitable tool
Development: 100 requests	74	16	10
Fresh: 100 requests	38	47	15

Table 1: Composition of the two suites. A capability can have more than one acceptable tool.

capability requests and unsupported operations.

Cases were authored during AI-assisted engineering, rather than sampled from customer traffic. Each records a query, groups of required tools, optional acceptable tools, forbidden tools, and a rationale. A group allows equivalent capabilities, such as reading a schema through `read` or `bash`. Requests include paraphrases, prerequisites, conditional actions, cross-integration tasks, and explicit instructions not to perform a related operation. Both suites and their labels are included in the research bundle.

4 Evaluation protocol

Each method receives the same case text and catalog. Direct methods search the request itself. Native methods receive a discovery instruction and generate their own search queries. Native callers are instructed to search before declaring a capability unavailable. Business tools have no execution function in this benchmark: the measured output is the set of definitions exposed to the caller.

Method	Configuration
Lexical	Existing substring scorer over names, descriptions, and schemas; top 5
Local BM25	Names, descriptions, and schemas; $k1 = 1.2$, $b = 0.75$; top 5
Jev	<code>typesafe-ai/jev</code> ; full descriptions; threshold 0.5; top 5
Anthropic BM25	Native BM25 search with <code>anthropic/claude-opus-5</code>
Anthropic regex	Native regex search with the same Anthropic caller
OpenAI native	Hosted search with <code>openai/gpt-5.6-sol</code>

Table 2: The six evaluated configurations. The local BM25 baseline is separate from Anthropic’s hosted implementation.

Native workflows allow up to three model steps and 2,048 output tokens per step. Their instruction requests at most three searches and preferably five tools per search; OpenAI also receives a three-tool-call limit. We score the union of discovered definitions across steps. The native five-tool preference is not a hard cap. Anthropic typically exposes about ten definitions, whereas direct methods return at most five. OpenAI uses individual deferred functions rather than namespace grouping [2]. These are comparisons of configured discovery workflows, with different exposure budgets.

Complete retrieval. A supported request is complete when the selected set contains at least one member of every required capability group. An unsupported request is complete only when no tools are returned. Extra tools do not invalidate completeness.

Precision. For a nonempty selected set, precision is the fraction labeled required or optional. An empty selection scores one for an unsupported request and zero for a supported request. Macro precision is the average over cases. We also report precision and completeness separately for supported requests.

Context exposure. We count unique selected tools and the canonical serialized bytes of their definitions. Forbidden-tool exposure records a mismatch with the request, not execution of an unauthorized

action.

We retain the first valid selection, including incorrect selections. Request validity is checked before scoring. The main six-method comparison uses the same 95 fresh cases with valid selections from every method; Appendix B gives all available denominators. Transport attempts are not retrieval decisions and do not enter the quality scores.

For uncertainty, we report Wilson intervals for complete-retrieval proportions and exploratory paired comparisons. Precision intervals use 10,000 paired bootstrap samples with a fixed seed. The paired binary test is exact McNemar. We do not adjust these exploratory comparisons for multiple testing.

5 Fresh evaluation results

Jev completes 89 of the 100 fresh requests with 80.4% macro precision. On the shared 95-case cohort, it completes 84 requests with 80.3% precision, compared with 77 and 64.4% for OpenAI native search and 79 and 15.3% for Anthropic BM25.

Method	Complete / 95	Macro precision	Correct no-tool	Mean tools
Jev	84/95	80.3%	7/14	1.83
OpenAI native	77/95	64.4%	1/14	2.23
Anthropic BM25	79/95	15.3%	0/14	9.69
Anthropic regex	56/95	11.8%	0/14	10.45
Local BM25	55/95	21.5%	0/14	5.00
Lexical	42/95	16.4%	0/14	5.00

Table 3: Matched comparison on 81 supported and 14 unsupported fresh requests.

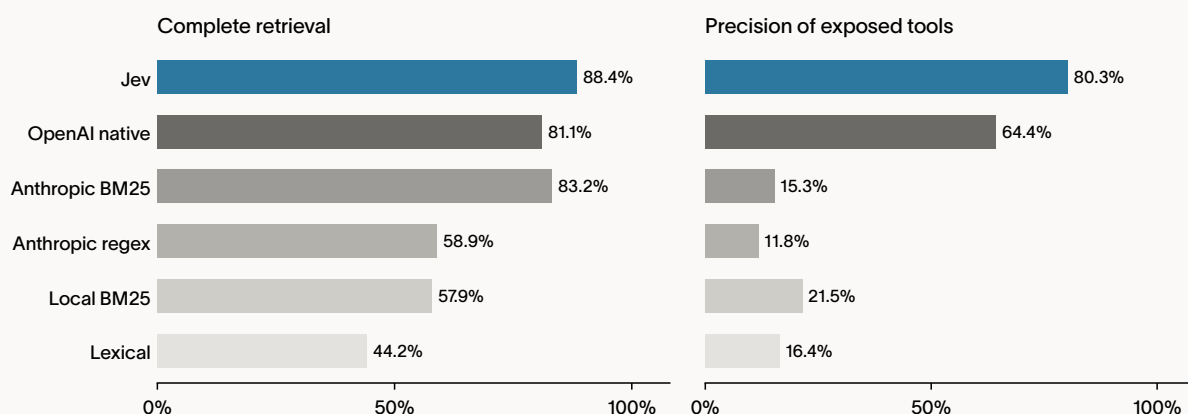


Figure 1: Complete retrieval and macro precision on the same 95 fresh requests.

The methods reach these totals in different ways. On the shared supported requests, Anthropic BM25 covers 79/81, Jev 77/81, and OpenAI 76/81. Jev's higher overall completeness comes from stronger abstention on unsupported requests. Anthropic BM25 achieves broad coverage while exposing substantially more tools, which lowers its precision under the exposure metric.

On all 100 fresh requests, Jev returns a complete selection with no irrelevant tools in 60 cases, compared with 46 for OpenAI. Its positive-case precision is 85.2%, compared with 75.5% for OpenAI. The

observed advantage is therefore a combination of selective exposure and abstention, rather than uniformly higher recall across every kind of request.

6 Where selection still needs work

The fresh suite makes the remaining errors visible. Jev covers every single-capability request, but misses part of four multi-capability requests and returns a related tool on seven unsupported requests.

Fresh request type	Jev complete	OpenAI complete	Local BM25 complete
Single capability	38/38	35/38	36/38
Multiple capabilities	43/47	45/47	19/47
No suitable tool	8/15	1/15	0/15

Table 4: All 100 fresh requests for three methods with complete valid output sets.

Two Jev misses omit a schema-reading prerequisite before editing a workflow or an email theme. Another selects file presentation but omits image inspection when both are requested. In a conditional Linear issue task, it mixes in Airtable tools and omits the required Linear comment from the final five-tool selection. These examples show why identifying a plausible main operation is insufficient: the entire required set must survive selection.

All seven unsupported-request misses expose a read tool related to the requested write operation. For example, a request to add a new Airtable column returns schema inspection even though the catalog cannot create that column. The caller may still explain the limitation correctly, but the search has loaded a tool that cannot fulfill the request. Our metric captures that unnecessary exposure.

Development and fresh results. Jev scored 100/100 with 86.3% precision on the development suite, compared with 89/100 and 80.4% on the fresh suite. The fresh set has more multi-capability and unsupported cases, so the difference reflects both new wording and a changed case mix. The frozen evaluation replaces the development result as the main estimate in this paper. The development records remain useful for reproducing the prompt and description work.

Uncertainty. Jev’s 89/100 complete-retrieval rate has a 95% Wilson interval of 81.4% to 93.7%. In the 100-case pairing with OpenAI, Jev alone completes 11 cases and OpenAI alone completes three (exact McNemar $p = 0.057$). The paired macro-precision difference is 15.25 percentage points, with a bootstrap interval of 7.88 to 23.03 points. Against Anthropic BM25 on 96 paired cases, the complete-retrieval comparison gives $p = 0.180$. This sample provides clearer evidence of a precision difference than of a completeness difference between the strongest native workflows and Jev.

The original development changes preserved full descriptions and narrowed relevance to direct operations and indispensable prerequisites. The fresh results suggest further work on prerequisite coverage, integration disambiguation, and abstention. We did not make those changes during this evaluation.

7 Context exposure and selection cost

Across all 100 fresh requests, Jev exposes an average of 1.87 tools and 2,676.58 bytes of definitions. This is 98.25% less definition data than the full 153,320-byte catalog. Figure 2 compares the common 95-case cohort. These measurements describe each discovery result; tools retained from earlier

searches accumulate during a conversation.

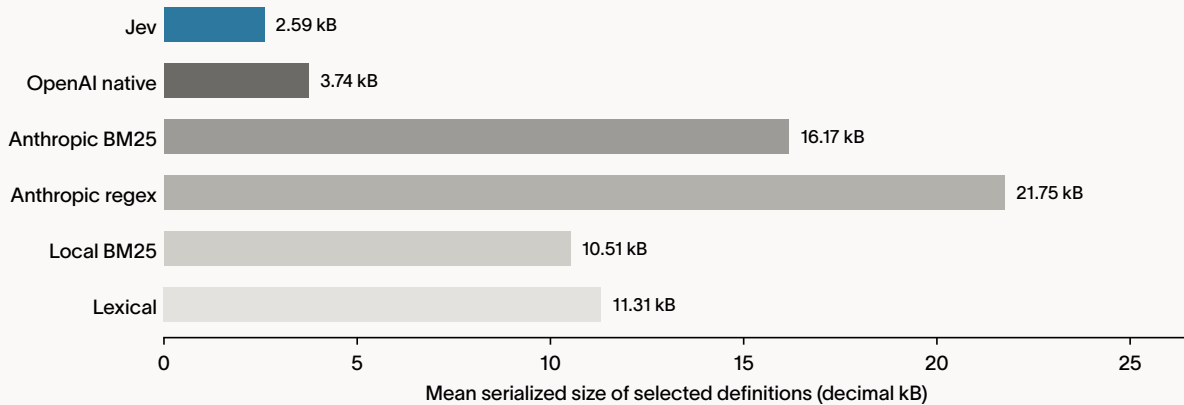


Figure 2: Mean definition exposure on the same 95 fresh requests. The complete catalog occupies 153.32 kB.

The successful fresh Jev evaluations report 3,735,642 input tokens, averaging 37,356.42 per search. At the recorded price of \$0.042 per million input tokens with no output-token charge, the selection step costs **\$0.1569 per 100 searches**, or **\$0.001569 per search** [4]. The rate was checked again on September 28 before completing the run.

The native workflows also incur caller-model input and output costs. Applying the recorded listed rates to their reported token usage gives the following selection-workflow totals:

Workflow	Listed token cost / 100 requests
Jev	\$0.157
OpenAI native	\$1.707
Anthropic BM25	\$10.488
Anthropic regex	\$13.301

Table 5: Token-priced discovery work in the fresh run. These totals cover different configured workflows and exclude any separate service fees. Local lexical and BM25 search make no inference calls.

Jev’s request includes the descriptions used to select tools. The downstream caller receives only the selected full definitions. These are separate costs: a reduction in caller context does not make the selector’s input free. The cost question is whether paying for that selection avoids enough repeated caller input over the conversation.

8 Conversation costs and caching

Let P_u , P_w , and P_r be the caller’s ordinary-input, cache-write, and cache-read prices per million tokens. Let U , W , and C be their token totals; Q the searches; c the selector cost per search; and H the additional uncached caller-input tokens used to request and receive a search.

$$\text{Eager} = \frac{P_u U_e + P_w W_e + P_r C_e}{10^6}. \quad (1)$$

$$\text{Lazy} = \frac{P_u (U_l + QH) + P_w W_l + P_r C_l}{10^6} + Qc. \quad (2)$$

The catalog contains 35,542 reference tokens using o200k_base, summed over canonical per-tool JSON. For a concrete scenario, assume 30 caller requests, six searches spaced every fifth turn, 2,000 distinct new tool-definition tokens per search, and $H = 500$. Loaded tools remain available. We use the fresh Jev fee and the September 22 caller-price snapshot: Opus 5 at \$5.00/\$6.25/\$0.50 per million ordinary/write/read tokens, and GPT-5.6 Sol at \$4.00/\$5.00/\$0.40 in the modeled context range.

Caller / turns	Cache treatment	Eager component	Lazy + search	Reduction
Opus 5 / 10	Stable prefix	\$0.382	\$0.046	87.9%
Opus 5 / 30	Stable prefix	\$0.737	\$0.198	73.1%
Opus 5 / 30	Rebuild at discovery	\$0.737	\$0.371	49.7%
Opus 5 / 30	Uncached	\$5.331	\$1.074	79.8%
GPT-5.6 Sol / 30	Stable prefix	\$0.590	\$0.161	72.8%
GPT-5.6 Sol / 30	Rebuild at discovery	\$0.590	\$0.299	49.4%
GPT-5.6 Sol / 30	Uncached	\$4.265	\$0.861	79.8%

Table 6: Modeled tool-input and search costs. The 10-turn example has two searches; 30-turn examples have six.

With a stable prefix, adding tools preserves the cached definitions already loaded. With rebuilding, each discovery rewrites the entire currently loaded tool set and later turns read that cache. Without caching, every caller request pays ordinary input rates. The model charges cache creation explicitly rather than treating every avoided token as an uncached saving.

The analysis sweeps 10, 30, and 60 turns, two rates of tool accumulation, and all three cache treatments. It also varies the caller-input and output overhead of discovery. Savings shrink when the conversation needs more of the catalog or when each search causes an expensive extra caller step. The modeled component is tool input plus discovery; it does not estimate the entire production conversation bill.

At Opus 5's ordinary-input price, the Jev fee alone equals about 314 input tokens. Including the assumed 500 caller-input tokens raises that to roughly 814 ordinary tokens, or 8,138 cached tokens at the \$0.50 rate. This explains why retention and caching matter alongside the selector's low per-search price.

9 Product implications and scope

A separate selector gives an integration a stable place to describe what it does while allowing the conversation model to change. As the product gains capabilities, the agent can discover tools as needed instead of expanding every caller prompt. The implementation includes opt-in deferral, permission-filtered catalogs, retained discovery state, and normal tool execution. The benchmark evaluates retrieval, not a production rollout of those features.

Our fresh evaluation measures new authored requests after configuration freeze. It remains a single product catalog with synthetic cases and labels created during AI-assisted engineering. The case mix is deliberate, not a production traffic estimate, and labels have not been independently adjudicated. Native search budgets and caller models differ; the study does not exhaustively tune every provider configuration. Model aliases are recorded, but weights are not pinned. End-to-end task success, larger catalogs, and real conversation cache behavior remain useful next measurements.

10 Related work

ToolLLM [5] studies tool use at much larger scale and includes retrieval in its pipeline. ToolRet [7] evaluates retrieval across thousands of queries and a broad tool pool. Our contribution is a smaller product-domain study connecting retrieval decisions to deferred loading, selective exposure, and conversation cost. Lu et al. [6] examine tool-documentation expansion; our use of full descriptions addresses the same practical concern that retrieval depends on how capabilities are documented.

Anthropic provides regex and BM25 search over deferred tools [1]. OpenAI provides hosted and client-executed search [2]. We evaluate configured native workflows alongside a selector that can serve different caller models. TypeSafe describes Jev as a structured evaluator [3]; here, its decisions determine which tool definitions become available to the agent.

11 Conclusion

Jev makes tool discovery practical as a separate, inexpensive decision in the agent harness. With a fixed configuration, it completes 89 of 100 fresh retrieval tasks at 80.4% precision and about \$0.00157 per search. Its strongest observed advantages are selective exposure and abstention; multi-capability prerequisites remain a source of misses. Modeling retained tools and cache behavior shows how a small selection cost can reduce repeated caller input.

For product teams, the broader lesson is to revisit decisions that were previously implemented as keyword rules or avoided by loading all available context. New decision models change the cost of making those choices explicitly. Tool search provides a concrete case where that possibility can be implemented, measured, and compared before broader adoption.

References

- [1] Anthropic. *Tool search tool*. Claude Platform documentation. Accessed September 21, 2026. <https://platform.claude.com/docs/en/agents-and-tools/tool-use/tool-search-tool>
- [2] OpenAI. *Tool search*. API documentation. Accessed September 21, 2026. <https://developers.openai.com/api/docs/guides/tools-tool-search>
- [3] TypeSafe AI. *Models*. Accessed September 21, 2026. <https://docs.typesafe.ai/models>
- [4] Vercel. *AI Gateway model catalog*. Price snapshots from September 21, 22, and 28, 2026 are included with the data. <https://ai-gateway.vercel.sh/v1/models>
- [5] Yujia Qin et al. *ToolLLM: Facilitating Large Language Models to Master 16000+ Real-world APIs*. 2023. arXiv:2307.16789. <https://arxiv.org/abs/2307.16789>
- [6] Xuan Lu et al. *Tools are under-documented: Simple Document Expansion Boosts Tool Retrieval*. 2025. arXiv:2510.22670. <https://arxiv.org/abs/2510.22670>
- [7] Zhengliang Shi et al. *Retrieval Models Aren't Tool-Savvy: Benchmarking Tool Retrieval for Large Language Models*. 2025. arXiv:2503.01763. <https://arxiv.org/abs/2503.01763>

A Final selection rule

For each tool name, the question asks whether it should be loaded to fulfill `state.query`. It selects direct requested operations and indispensable prerequisites, honors entity, integration, direction, and read/write intent, and excludes rediscovery when an identifier is supplied. Merely related tools, optional investigation, and hypothetical alternate workflows are excluded. Query and catalog text are treated as data rather than instructions that change the decision rule. Exact question text, criteria, validation, threshold, and sorting are preserved in `implementation/jev.ts`.

B Full-suite results

The main table uses the common 95-case cohort. The table below retains each method's full set of valid fresh selections. Four Anthropic BM25 outputs and one regex output did not produce scorable selections; those cases are omitted from every method in the matched comparison.

Method	Complete / scored	Macro precision	Correct no-tool	Mean tools
Jev	89/100	80.4%	8/15	1.87
OpenAI native	81/100	65.2%	1/15	2.23
Anthropic BM25	79/96	15.1%	0/15	9.70
Anthropic regex	58/99	12.0%	0/14	10.49
Local BM25	55/100	21.4%	0/15	5.00
Lexical	42/100	16.2%	0/15	5.00

Table B1: All available fresh-suite selections.

The development suite is reported separately because its cases informed the configuration:

Method	Complete / scored	Macro precision	Correct no-tool
Jev	100/100	86.3%	10/10
OpenAI native	96/100	66.9%	6/10
Anthropic BM25	87/98	13.9%	1/10
Anthropic regex	70/100	11.1%	1/10
Local BM25	69/100	18.4%	0/10
Lexical	57/100	15.8%	0/10

Table B2: Development-suite results. These are not pooled into the fresh evaluation.

C Reproduction

The bundle contains both case sets, the catalog, saved selections and scores, prices, implementation snapshots, and deterministic analysis scripts. `analyze-fresh.py` checks the sealed case hash, verifies recorded completeness, precision, and definition sizes, and regenerates the fresh metrics, paired comparisons, and cost scenarios. `figures-fresh.py` creates the publication figures. `analyze.py` and `sensitivity.py` reproduce development analyses. `build.py` renders the paper from this Markdown source.

The fresh cases were sealed on September 22, 2026 after selector freeze. Local and native evaluations ran on September 22; Jev evaluations completed on September 28 with the same selector. A recorded continuation amendment changes only spending-ledger locking. The fresh case SHA-256 begins 83bdcfff73194f1d; full file hashes and configured model IDs are preserved in the manifests. These dates describe one evaluation per case and method, not repeated attempts to choose a better relevance result.