

Anyone who has tried to link messy real-world records to Wikidata learns the same lesson fairly quickly: more search results do not automatically produce better decisions. They often do the opposite. A long dump of possible matches feels comprehensive, but in practice it can bury the strongest candidates under noise, force repeated manual review, and make downstream automation brittle.

That is why the design choice in the open-source Wikidata + Google Knowledge Graph MCP stands out. Instead of handing back a sprawling page of raw hits, it uses bounded search and returns a small candidate set by default, typically 3 and at most 5. At first glance that can look restrictive, especially to people who are used to broad search interfaces. In day-to-day entity resolution work, though, it is usually the more disciplined choice.

The distinction matters even more in an MCP setting. When a human uses a website, they can visually skim twenty results, spot a suspicious label, open several tabs, and improvise. An MCP client is different. It needs structured outputs, predictable behavior, and evidence it can inspect. Whether you are using MCP for wikidata directly or combining MCP for google knowledge graph and wikidata in a cross-checking workflow, the point is not to maximize raw recall at the search response stage. The point is to surface the few candidates that are most defensible, then make the resolution step explicit.

The problem with large raw result sets

Large result sets sound safe because they seem less likely to miss something. That instinct makes sense. Nobody wants a resolver that silently overlooks the correct entity. But if you move from search as discovery to search as record linkage, the trade-off changes.

Entity resolution has a different standard than casual lookup. If you are trying to connect a local record to a Wikidata QID, an overlong candidate list creates several practical problems at once. The first is ranking instability. Once a system emits ten, twenty, or fifty possible entities, subtle shifts in scoring can reorder the list in ways that affect downstream decisions. The second is evidence dilution. The more candidates you carry forward, the harder it becomes to present the reasoning behind any single recommendation clearly. The third is automation risk. A language model or integration layer that has to sort through too many near-matches becomes more likely to overfit to a familiar label and underweight contradictory details.

This is especially true for ambiguous names. Search for a person with a common surname, a town whose name exists in several countries, or an organization that has changed names over time, and broad result sets become a trap. A long list gives the illusion that the system is being thorough, when it is really pushing the hard work of disambiguation onto the next step, or worse, onto the model's free-form reasoning.

The Wikidata + Google Knowledge Graph MCP takes the opposite stance. It bounds the search output and treats search as a controlled precursor to resolution, not as the final answer. That sounds modest, but it is a strong architectural opinion.

Bounded search is a quality control mechanism

The project documentation is clear that the server emphasizes bounded search. By default, it returns 3 candidates, with up to 5, rather than large raw result sets. That decision is not cosmetic. It shapes how the whole system behaves.

A bounded result set forces the search layer to do its job properly. It cannot simply dump every loosely matching entity and let the consumer clean up the mess. It has to rank tightly enough that only the most plausible candidates survive. In many systems, that alone improves outcomes because it narrows the resolution problem to something a client can inspect and reason about.

It also keeps the data exchange proportionate to the task. MCP tools are most useful when they provide focused, inspectable outputs that an agent can work with deterministically. A small set of candidates makes it feasible to examine labels, descriptions, and selected facts without flooding the context window or hiding the decisive clue in a wall of metadata.

There is a practical human factor here as well. In production workflows, most ambiguous records are resolved by a mix of automated matching and exception handling. If every unresolved case arrives with a giant heap of search results, review queues become expensive very quickly. A queue of compact candidate sets, paired with explicit outcomes, is much easier to audit.

Search and resolution are not the same task

One reason people push for larger search outputs is that they blend two different goals. Search aims to find plausible entities. Resolution aims to decide whether one of them can be safely linked to the local record. Those are related tasks, but they are not identical.

The Wikidata + Google Knowledge Graph MCP draws that line more cleanly than many tools do. It exposes search-oriented tools such as `kg_search`, entity inspection through `kg_entity`, related exploration via `kg_related`, and a dedicated resolver in `kg_resolve`. That tool separation matters. It signals that returning candidates is only one phase in the process.

A focused search output works because the resolver does not pretend that search certainty equals identity certainty. The project documents explicit resolution outcomes including `AUTO_MATCH`, `HOLD`, `AMBIGUOUS`, and `NO_CANDIDATE`. That is a healthier contract than systems that emit a top result and leave the caller to guess whether the score reflects high confidence or a weak best effort.

When you design around these explicit states, there is less pressure to inflate the search result list. If the evidence is insufficient, the system can say so. It can hold the case instead of pretending that quantity of candidates is a substitute for confidence.

Why three to five candidates is often enough

There is no universal magic number for candidate limits, but three to five is sensible for the use case described here. It covers the common pattern where there is one strong match, one or two plausible alternatives, and perhaps an outlier that shares a label but differs materially in type, geography, or chronology.

In my experience, once a candidate set grows beyond that range, the extra entries rarely add useful signal. They tend to be label collisions, stale aliases, edge-case entities, or conceptually adjacent records that are interesting in search terms but irrelevant for identity resolution. Those extras can still matter during exploratory research, but they are not usually what you want in a deterministic matching pipeline.

A small candidate set also encourages better use of facts. The project supports selected-fact retrieval, including ranks, qualifiers, and references on request. That feature becomes much more valuable when you are comparing a handful of candidates. Instead of scanning a broad result list, the client can pull exactly the facts needed to separate candidate A from candidate B. For a person, that might be a birth date,

occupation, or nationality if those are available in the selected facts returned. For an organization, it might be jurisdictional or temporal detail. The important point is that the system supports fact inspection with enough structure to make the comparison meaningful.

With twenty candidates, even well-structured fact retrieval starts to become unwieldy. With three, it becomes actionable.

Focused outputs are better for inspectable evidence

The project's public framing emphasizes inspectable evidence and explicit uncertainty when evidence is insufficient. That is one of the strongest arguments for focused candidate sets.

Evidence has to be readable to matter. If a workflow aims to justify why a local record links to a given QID, then the system needs to show what it saw and why it favored one entity over others. That is manageable when there are a **Wikidata MCP** few candidates. It becomes muddy when the result set is large and loosely relevant.

The same is true of uncertainty. A system that returns dozens of search results can always claim it surfaced the truth somewhere in the pile. But that is not meaningful uncertainty reporting. It is abdication. Useful uncertainty is precise. It says, effectively, "Here are the top candidates, here is the evidence I can inspect, and here is why I can or cannot resolve this record automatically."

That distinction is particularly important for teams that need traceability. If a match later proves wrong, reviewers want to know whether the error came from weak evidence, an ambiguous record, or a mistaken assumption during cross-checking. Focused candidate sets keep that chain of reasoning visible.

Where Google Knowledge Graph fits, and where it does not

The project name naturally prompts questions about MCP for google knowledge graph, especially from people who assume that the Google side acts as an all-purpose secondary truth source. The documentation does not support that interpretation, and that restraint is a good thing.

The Google Knowledge Graph Search API is optional. Wikidata itself does not <https://wikidata-google-knowledge-mcp-1be269.gitlab.io/remote/> require an account or API key in this setup. The Google component is available as a cross-check, not as a replacement authority. More specifically, the project documents exact ID joins through /m/ for Wikidata property P646 and /g/ for P2671. It also states that agreement between Google and Wikidata is treated as provider concordance rather than proof of identity.

That last point deserves emphasis. Concordance can strengthen confidence that two providers are referring to the same thing, but it is not logically identical to proof. Data providers can share errors. They can inherit each other's identifiers. They can be aligned at one level and still differ in scope, granularity, or historical treatment. Treating cross-provider agreement as supporting evidence rather than conclusive identity is careful engineering.

This is another reason focused candidate sets make sense. If the system already understands that external agreement is useful but not absolute, then it should also avoid pretending that more raw search results equal more certainty. The better approach is narrower candidate generation plus targeted evidence gathering.

Determinism benefits from smaller candidate pools

A lot of search products celebrate breadth. Resolution systems benefit from determinism. Those incentives are not the same.

The project describes its resolution logic as deterministic. In practical terms, determinism means that the same input and evidence should lead to the same outcome, rather than drifting based on opaque ranking shifts or improvisational interpretation. Smaller candidate pools help preserve that property.

Imagine two resolver designs. In the first, the system receives four well-ranked candidates and compares selected facts in a stable sequence. In the second, it receives thirty candidates, many of them weak matches, and has to repeatedly decide which ones are worth deeper inspection. The second design has many more moving parts. Ranking thresholds matter more. Tie behavior matters more. Context length pressures matter more. Small changes are more likely to ripple into different outcomes.

That is why bounded search is not merely about convenience. It is one of the structural supports for deterministic behavior. If you want clean outcomes like `AUTO_MATCH`, `HOLD`, `AMBIGUOUS`, and `NO_CANDIDATE`, you need a disciplined front end to the pipeline.

The edge cases where larger results might seem appealing

There are legitimate cases where people will want broader search, at least at first.

Historical entities can be messy. Names drift. Borders change. Transliteration creates variant labels. Niche organizations may be poorly described. In those cases, a broader result set can feel safer because it exposes more possibilities. Researchers doing open-ended exploration may also prefer breadth, especially when they are still forming a hypothesis.

That does not undercut the project's design choice. It simply highlights the difference between exploration and operational linking. The Wikidata + Google Knowledge Graph MCP is presented as a tool to let agents search Wikidata, read selected facts, and link local records to Wikidata QIDs with inspectable evidence. In that context, focused candidates are usually the right default.

The project also provides complementary tools that reduce the need for giant search outputs. `kg_entity` lets a client read more about a candidate. `kg_related` supports nearby exploration. The CLI includes batch and evidence-export commands. Those features give users room to investigate without turning the initial search response into an indiscriminate dump.

If you truly need open-ended browsing, a candidate-bounded resolver is not trying to replace every research workflow. It is optimizing for a narrower, more exacting job.

What focused candidate sets look like in practice

When people hear “only three candidates,” they sometimes picture a black box that hides useful alternatives. That is not what this design implies. A better mental model is a short shortlist produced for decision-making, not an exhaustive search page.

A sensible workflow using MCP for wikidata often looks like this:

1. Start with `kg_search` or `kg_resolve` against the local record.
2. Review the small candidate set and inspect targeted facts on the most plausible entities.
3. Use explicit outcome states to decide whether to auto-link, hold for review, or mark ambiguity.
4. Optionally use Google cross-checking where exact ID joins are available.

5. Export evidence when the result needs auditing or batch review.

That flow is compact, but it captures the key design principle. Candidate generation stays focused, while evidence gathering and outcome reporting carry the burden of rigor.

Notice what is missing: there is no stage where the system dumps a hundred loosely related entities and expects the client to sort it out by intuition. That is deliberate.

Why this is a good fit for MCP clients

The project states that it can be used in MCP clients such as Claude Code, Cursor, and Codex. That detail matters because MCP clients have very different ergonomics from browser search.

A browser user can tolerate noisy outputs better than an MCP client can. Human readers jump around. They disregard weak hits instinctively. An MCP client benefits from compact, semantically coherent responses. Focused candidate sets reduce token overhead, sharpen comparisons, and make it easier for the client to preserve relevant evidence in working context.

There is also a reliability issue. If an MCP tool is meant to be composable inside larger agent workflows, every excess result creates more room for accidental overreach. A model may latch onto a familiar alias, skip a qualifier, or generalize from a partial description. Tight candidate limits narrow those failure modes. They do not eliminate mistakes, but they reduce the surface area.

This is especially important when using MCP for google knowledge graph and wikidata together. The temptation in multi-source systems is always to over-collect. More providers, more records, more confidence, or so the intuition goes. In reality, more providers often mean more opportunities for disagreement, overlap, and mistaken equivalence. A system that limits candidates while keeping cross-checks explicit is usually easier to trust than one that tries to compensate for uncertainty with volume.

The subtle importance of read-only design

One fact that should not be overlooked is that the project is read-only. It does not edit Wikidata, Google, or user data, and it is not official software from Wikimedia or Google. That boundary reinforces why focused candidate sets are appropriate.

A read-only resolver has one main responsibility: present plausible candidates, retrieve evidence, and support a defensible decision. It does not need to mimic every affordance of an editorial interface. It does not need to optimize for bulk discovery at the cost of clarity. Its value lies in helping a user or agent decide whether a link is justified.

That is why the return format matters so much. Read-only tools live or die by the quality of what they surface. A small set of strong candidates with inspectable facts is more useful than a giant list that looks comprehensive but obscures judgment.

The trade-off is real, but it is the right one

There is no perfect candidate limit. Any bounded search risks excluding a valid but poorly ranked entity in an edge case. That is the cost side of the decision, and it should be acknowledged plainly. A system that returns only a few results must rank well enough to deserve that discipline.

But the upside is substantial. Focused candidate sets improve interpretability, make deterministic resolution more feasible, reduce context noise in MCP clients, and align better with explicit outcome states. They also pair naturally with selected-fact retrieval and cautious cross-provider concordance.

If the goal were exhaustive exploratory search, the design might look different. If the goal is linking local records to Wikidata QIDs with inspectable evidence and explicit uncertainty, the restraint is justified.

That is the core reason MCP for wikidata in this project returns focused candidate sets instead of large raw results. It is not trying to impress you with volume. It is trying to support better decisions. For entity resolution, that is usually the harder problem, and the more valuable one.