

When people first hear "search" in the context of Wikidata, they often picture a wide funnel: type a label, get back a long stack of possible entities, and let a human sort it out. That approach works for exploratory browsing. It breaks down fast when the user is not a person clicking through results, but an AI agent trying to make a careful decision inside a larger workflow.

That is where bounded search matters.

In the case of [read more](#) the open source "Wikidata + Google Knowledge Graph MCP" server and CLI, bounded search is not a cosmetic detail. It is part of the operating model. The project is designed to help AI agents search Wikidata, inspect selected facts, and link local records to Wikidata QIDs with evidence the user can inspect. It also makes uncertainty explicit when the evidence is not good enough. Those goals only hold together if search stays disciplined.

The project's default behavior reflects that discipline. Instead of dumping a large raw result set, it returns a small candidate set, three by default and up to five. That sounds simple, almost obvious. In practice, it changes how an MCP client reasons, how evidence gets reviewed, and how errors are contained.

Why "bounded" is doing real work here

A large result set creates a false sense of completeness. It can look thorough while actually making resolution less reliable. An AI system presented with dozens of near matches often starts leaning on shallow signals: token overlap, popularity, recency, or whatever happens to dominate the ranking. The wider the result set, the easier it becomes to rationalize the wrong match.

A bounded set forces a different behavior. It says, in effect, "Here are the strongest candidates we can justify exposing right now. If none are good enough, stop." That is a healthier pattern for entity resolution, especially when the downstream action is to attach a persistent identifier to a local record.

I have seen this distinction matter in ordinary cataloging and reconciliation work. A human reviewer with 40 results often skims and guesses. A reviewer with three strong candidates reads. The same holds for an agent. Constraining the choice set reduces the temptation to overfit weak evidence.

That is especially important in Wikidata, where ambiguity is normal. Many people share names. Organizations rebrand. Works appear in multiple editions. Places inherit labels from older jurisdictions. If your search layer encourages broad fishing, you push the ambiguity problem downstream. If your search layer bounds the candidate set and pairs it with explicit resolution outcomes, you handle ambiguity where it belongs.

MCP changes the audience for search

Wikidata's own MCP context is already about giving language models standardized tools to explore and query Wikidata programmatically. The moment you put search behind MCP, the user is no longer just a researcher at a keyboard. It may be Claude Code, Cursor, Codex, or another MCP client acting on behalf of a user. That matters because the search output is now machine-consumed.

Machines do not get tired, but they do drift. They can treat a ranked list as implicit permission to choose something. They can confuse "top result" with "verified match." A bounded design counters that tendency by keeping the output narrow and manageable.

The "Wikidata + Google Knowledge Graph MCP" project leans into this. It is read-only. It does not edit Wikidata, Google, or user data. It is not official software from Wikimedia or Google. It is also explicit that any Google and Wikidata agreement is provider concordance, not proof of identity. Those boundaries are technical, but they are also philosophical. The server does not pretend that search equals truth. It offers evidence, selected facts, and a small number of candidates, then makes room for uncertainty.

That is a much better fit for MCP than the classic "return everything and let the caller decide" pattern.

What bounded search looks like in this project

The project documents a set of MCP tools, including `kg_search`, `kg_entity`, `kg_related`, `kg_resolve`, and `kg_status`. The CLI adds batch and evidence export commands. Even without speculating about internal implementation, you can see the intended shape of work.

A client starts with a search request. Instead of receiving a sprawling result page, it gets a bounded set of likely candidates. If one candidate looks promising, the client can fetch selected facts. Those facts can include ranks, qualifiers, and references when requested. If the task is entity resolution, the client can use deterministic resolution logic with explicit outcomes such as `AUTO_MATCH`, `HOLD`, `AMBIGUOUS`, and `NO_CANDIDATE`.

That sequence is worth dwelling on. Search is not the final answer. Search is the gatekeeper for the evidence-gathering stage. Bounded search makes the next step more meaningful because the agent is not trying to compare fifteen barely relevant entities. It is inspecting a handful of plausible ones.

The difference is practical. Suppose a local record contains a person name and a birth year. With a large unbounded search, the agent may see many entities with the same label and weakly infer a match based on prominence or partial overlap. With a bounded search, the system narrows the field, and the agent can then inspect the selected facts that matter most for disambiguation, perhaps the birth date, occupation, or other requested statements with qualifiers and references. If that still does not settle the case, the resolution outcome should remain uncertain.

That is a better failure mode than a confident mistake.

Small result sets improve reviewability

One of the less glamorous advantages of bounded search is auditability. If a local record was linked to the wrong QID, someone needs to understand why. That gets much harder when the initial search produced a large result pool and the client sifted through it with opaque heuristics.

A bounded set makes the chain of reasoning easier to reconstruct. Reviewers can look at the candidate list, inspect the evidence pulled from `kg_entity`, and understand why the resolver returned `AUTO_MATCH` or why it held the record for review. That is valuable whether you are processing ten records or ten thousand.

It also aligns with the project's promise of inspectable evidence. The phrase sounds modest, but it addresses a real operational problem. Teams often discover too late that their entity links cannot be defended. Someone asks, "Why did the system attach this identifier?" And the answer is a shrug wrapped in confidence scores. A bounded search plus selected-fact retrieval produces a far cleaner paper trail.

In reconciliation work, the most useful evidence is rarely the most abundant. It is the most discriminating. Three candidates with targeted facts are more useful than thirty candidates with shallow metadata.

Bounded search is not the same as limited capability

A common misunderstanding is that a tight candidate cap means the system is weak or simplistic. Not necessarily. Often it means the system has strong opinions about what a safe search interface should provide.

This project still supports several kinds of activity. You can search, inspect entities, look at related items, resolve candidate matches, check status, and work through the CLI in batch with evidence export. It can also optionally cross-check against the Google Knowledge Graph Search API. What changes is not the existence of these capabilities, but the shape of the output and the standards for acting on it.

That distinction matters if you are evaluating MCP for Wikidata in a production setting. A broad search interface can feel more flexible because it exposes more raw material. Yet flexibility often hides transfer of burden. The burden simply moves from the tool into your prompting, your resolver logic, your review queues, and your cleanup efforts after bad links appear.

Bounded search moves some of that burden back into the design of the interface. That is usually a good trade when the goal is reliable linking rather than free-form exploration.

The role of deterministic outcomes

The explicit resolution states in this project are one of its strongest signals of maturity. `AUTO_MATCH`, `HOLD`, `AMBIGUOUS`, and `NO_CANDIDATE` tell you the resolver is not pretending every search must end in a match.

That sounds basic, but it is where many systems go wrong. They treat entity resolution as a classification problem with a mandatory winner. Real data does not cooperate. Some records are too sparse. Some are internally inconsistent. Some correspond to entities that are not represented in the expected way. Some are genuinely ambiguous.

Bounded search works best when paired with a resolution vocabulary that can say no. In fact, the cap on candidates makes those outcomes more honest. If no convincing match is present among the strongest few candidates, that does not mean "search harder until something looks close enough." It means the system should stop, surface uncertainty, and let the workflow decide what happens next.

Here the design is doing more than limiting output. It is shaping operator behavior. Teams that adopt tools like this usually find that a visible `HOLD` category saves them from a mountain of silent errors. It creates a place for undecidable cases. Every experienced data curator knows you need that place.

How Google cross-check fits without overreaching

The project's optional Google cross-check is a subtle but important feature. It documents exact ID joins using `/m/` for Wikidata property `P646` and `/g/` for `P2671`. Just as important, it states that agreement between Google and Wikidata should be treated as provider concordance, not proof of identity.

That is the right level of restraint.

People are often tempted to use a second data source as a truth oracle. If both providers point in the same direction, confidence rises. Fair enough. But concordance is not identity proof. Shared IDs and aligned records are strong signals, not metaphysical guarantees. Data providers can inherit errors, lag updates, or model the same thing differently.

In a system built around bounded search, this optional cross-check works best as supporting evidence, not as a way to bulldoze uncertainty. If the bounded candidate set surfaces a plausible Wikidata item, and exact identifier joins align with Google, that may strengthen the case. If the candidate set remains ambiguous, concordance should not magically erase the ambiguity.

That point deserves emphasis because it shows the difference between careful MCP for Google Knowledge Graph and Wikidata, and a looser integration that treats any cross-provider overlap as validation. The more disciplined approach is slower in edge cases, but far safer.

Selected facts matter more than exhaustive dumps

Another reason bounded search works here is the project's support for selected-fact retrieval, including ranks, qualifiers, and references on request. This is a much better fit for agent workflows than a maximal entity dump.

When an agent is resolving a record, it usually does not need every statement attached to an entity. It needs the few facts that separate likely matches from near misses. Qualifiers can make or break that judgment. So can rank. References matter because they let a reviewer inspect the basis of a claim rather than accepting a bare assertion.

This is the quiet strength of the design. Bounded search narrows the set of entities worth considering. Selected-fact retrieval deepens the evidence only where needed. Together they create a workflow that is both economical and inspectable.

I would go further and say that this pattern is better aligned with how expert humans work. Experienced researchers do not read entire records for every possible match. They shortlist, then inspect the discriminating facts. The MCP interface here appears to formalize that habit.

Where bounded search can feel restrictive

No design choice is free. Bounded search can frustrate users who are used to open-ended exploration. If you are investigating a messy topic with uncertain labels, a cap of three candidates by default may feel tight. There will be cases where the right answer is not in the first few results, or where a broader sweep would help a specialist notice a pattern.

That tension is real. Search for discovery and search for resolution are not the same activity. Discovery tolerates breadth because the human is intentionally exploring. Resolution needs control because the system is deciding whether to attach an identifier.

The project seems to choose resolution discipline over exploratory abundance. Given its stated purpose, that is sensible. It aims to help agents link local records to QIDs with inspectable evidence, not to replicate every kind of browsing behavior a human researcher might want.

For teams adopting it, the lesson is simple: use bounded search for matching workflows, and do not expect it to behave like a general-purpose exploratory interface. If your task is archival research, broad querying methods may still have a place elsewhere in your stack.

Practical cases where bounded search shines

The clearest wins usually appear in records that are common, repetitive, and easy to get subtly wrong. A local database might hold names of people, organizations, or works with only a few descriptive fields. In

those situations, large result sets are not extra help. They are extra noise.

Bounded search is especially useful when you care about these operational outcomes:

- fewer low-quality automatic links
- easier human review of uncertain cases
- clearer evidence trails for every accepted match
- deterministic handling of records that should not be matched
- safer use of optional cross-provider checks

Those are not glamorous improvements, but they are the ones that keep data pipelines healthy six months later.

I have found that teams often underestimate the cost of a wrong identifier. A bad link can propagate into reporting, enrichment, deduplication, and search relevance. Fixing one mistaken QID is easy. Finding all the records contaminated by the same faulty matching habit is harder. Bounded search is one way to make that habit less likely to form.

Why this matters for MCP for Wikidata

If you look at the broader conversation around MCP for Wikidata, the most interesting question is not whether models can query a knowledge source. They can. The harder question is whether the interface encourages disciplined use of that source.

A model with unrestricted access to broad search and rich query tools may appear powerful while still behaving recklessly in entity resolution. A more constrained interface can produce better practical outcomes because it narrows the room for overconfident guessing.

That is why bounded search deserves attention. It is not just a UI preference, and not just a performance tweak. It is a policy decision embedded in the interface. It tells the client that candidate generation should be selective, that evidence should be inspectable, and that uncertainty must remain visible.

For anyone exploring MCP for google knowledge graph and wikidata, this project offers a concrete example of that philosophy. The combination of small candidate sets, deterministic resolver outcomes, optional exact-ID cross-checking, and selected-fact inspection forms a coherent pattern. Each part supports the others.

The deeper design principle

Underneath the technical details is a principle that experienced data people learn sooner or later: not every plausible match should be turned into an actual link.

Systems get into trouble when they collapse possibility into identity. Search returns an item, therefore the item must be right. Two providers align, therefore the entity is proven. A label overlaps, therefore the record is resolved. Bounded search pushes against that reflex by reducing the volume of candidates and making each candidate carry more evidentiary weight.

That principle also explains why the project's read-only posture matters. Because it does not edit Wikidata, Google, or user data, it can focus entirely on retrieval, comparison, and evidence. It is a useful boundary. The moment a tool both suggests and writes links, the quality bar rises dramatically. A bounded, inspectable, read-only design is a sensible place to start.

What to watch for in real use

If you evaluate this kind of server in practice, pay attention less to how often it finds something, and more to how it behaves when it should not decide. The best signal of quality is not a high match count. It is graceful uncertainty.

Watch whether the candidate cap keeps the review surface manageable. Watch whether selected facts are enough to separate strong matches from tempting decoys. Watch whether `AMBIGUOUS` and `NO_CANDIDATE` appear often enough to suggest honesty rather than forced confidence. If you add the optional Google cross-check, watch whether your team treats concordance as support instead of proof.

Those are the habits that turn a useful MCP tool into a trustworthy one.

For people specifically considering MCP for wikidata or comparing it with an MCP for google knowledge graph workflow, the lesson is straightforward. Search is not only about recall. In linking and resolution tasks, search quality also depends on boundedness, inspectability, and explicit stopping points.

The best systems know when to stop offering options and start demanding evidence. This project's approach to bounded search suggests it understands that distinction well.