ML LABS INTELLIGENCE

Building Natural Language Search for Operational Records

Domain experts needed to search a production AI platform's worklist in plain language. ML LABS built the query parser, search engine, and mobile interface.

CASE STUDY

Author Omar Trejo

Date 2026-03-05

A multi-tenant AI platform had a growing usability problem: domain experts needed to find specific records across a large operational worklist, but the only way to search was through a structured filter UI with dozens of fields and dropdown combinations. Experts think in the language of their domain — "unconfirmed AI findings from last week" or "abnormal impressions for Site 12" — not in filter combinations. The gap between how they think and how the system expected them to search was slowing down the people who needed data most urgently.

ML LABS built a natural language search capability that let users query the worklist in plain language. The system parses domain-specific intent into structured queries, searches across both raw operational records and AI-generated outputs, and returns results fast enough for real-time use on desktop and mobile.

The Problem

The worklist contained operational records annotated with AI-generated outputs: algorithmic classifications, interpretive analyses, and AI impressions produced by multiple model providers. Searching this data required combining filters across both the raw record fields (site, date, status, assigned expert) and the AI output fields (classification result, impression text, confirmation status). The structured filter UI exposed these as separate controls, and combining them required knowing which fields existed and how they interacted.

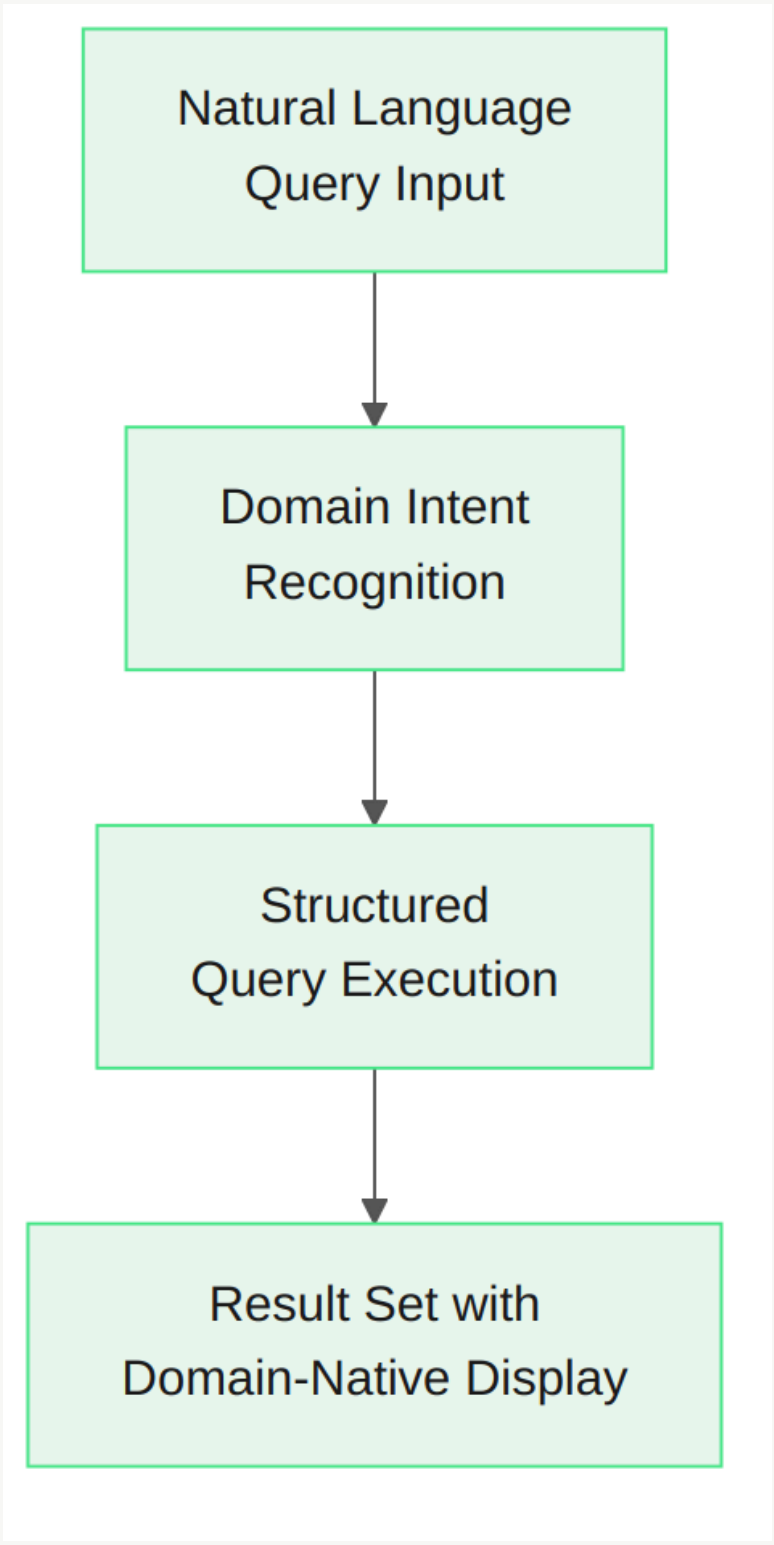
- Domain experts searched in natural language but the UI only accepted structured filter combinations
- AI-generated outputs (impressions, classifications, interpretive analyses) needed to be searchable alongside raw records
- Query performance had to support real-time use — results within the operational workflow, not batch retrieval

- Mobile users needed identical search capability without the screen space for complex filter UIs
- Name searches needed to work on first name, last name, or both without requiring the user to specify which

Natural Language Query Parser

The core of the search system is a constrained intent parser that maps natural language queries to a known set of domain entities and structured filters. This is not general-purpose text-to-SQL — the parser recognizes entities specific to the operational domain and translates them into executable queries.

When a user types "abnormal AI impressions for Site 12 this month," the system understands the intent — an AI classification status, a site identifier, and a date range — and returns the right results. The parser handles ambiguity through explicit resolution rules developed against real user queries collected from domain experts during the build, not through probabilistic guessing.


Natural Language Query Input
Domain Intent Recognition
Structured Query Execution
Result Set with Domain-Native Display

> The parser's reliability comes from constraint, not sophistication. It recognizes every entity that maps to a filterable field in the operational database — and nothing else. Bounded vocabulary means predictable results.

Searching AI-Generated Outputs

The worklist contained two layers of searchable data: raw operational records and AI-generated outputs produced by the platform's inference pipeline. AI impressions, interpretive analyses, and model-generated labels all needed to be searchable in the same interface as the underlying records.

ML LABS built unified search across both layers. A query for "abnormal AI impression" searches AI outputs. A query for "Site 12 records from last week" searches primary records. A query combining both returns the intersection — and the user never needs to know which data layer is being queried.

AI-generated outputs were treated as first-class searchable content, not bolted on as secondary features. When a model provider produced a new output type, the search system expanded to include it as a configuration change rather than a code change.

Query Performance

Operational search has a hard latency requirement. Domain experts search within their workflow — they type a query, scan results, and act. If results take more than a second, the search system disrupts the workflow rather than supporting it.

ML LABS optimized query execution to deliver sub-second results at production data volumes. The optimization was driven by actual query patterns collected during testing, not hypothetical usage — ensuring the most common searches hit the fastest paths. Response payloads were trimmed to only the fields required for display, and large result sets did not block initial rendering.

Mobile Search

Domain experts increasingly worked from tablets and phones. The search interface needed to deliver identical functionality on mobile without the screen space for the desktop filter UI — making natural language search not just a convenience but the primary input method on smaller screens.

ML LABS built mobile search using the same query engine as desktop. The adaptation was entirely in the presentation: touch-friendly result cards, appropriately sized result sets, and simplified disambiguation for ambiguous queries. The search behavior remained identical across platforms — same parser, same results, same performance.

Boundary Conditions

Natural language search works when the domain vocabulary is bounded and query patterns are predictable. It struggles when the underlying data quality is poor — inconsistent naming conventions, missing fields, or duplicate records produce unreliable results regardless of how well the intent parser works. The first investment in those environments is upstream data quality, not a better search interface.

It also reaches limits when queries target genuinely unstructured content. Searching freeform narrative notes or unstructured text fields is a full-text ranking problem, not a structured filtering problem. Attempting to force NLP parsing onto unstructured content produces worse results than keyword search. Getting the search architecture right on the first build matters more than in most software work — search systems that launch with incorrect index coverage or misaligned entity mapping erode user trust in ways that are difficult to reverse.

Results

Domain experts search the production worklist in natural language instead of navigating complex filter UIs. AI-generated outputs — impressions, classifications, interpretive analyses — are searchable alongside raw operational records through a unified interface. Query performance supports real-time use at production data volumes. Mobile search works identically to desktop, with the natural language parser serving as the primary input method on smaller screens.

The system handles the queries domain experts actually ask: searches by name (first, last, or both), searches by AI classification status, searches combining site filters with date ranges and AI output criteria, and generalized model label searches across the full worklist. Filter test suites validate query parsing against real-world query patterns, and the search pipeline serves as the foundation for future query capabilities without architectural changes.

First Steps

1. **Collect real queries from domain experts before building the parser.** Log the natural language searches users attempt across at least one operational workflow. The distribution of actual queries — which filters dominate, which

ambiguities recur, which data sources get combined — determines the parser's scope and the indexing strategy.

2. **Build the structured filter API first, then add the NLP layer on top.** The filter API is the stable foundation that accepts explicit parameters and returns results. The NLP parser translates natural language into those parameters. Building in this order means search works via explicit filters before the NLP layer is ready.
3. **Measure latency at the 95th percentile under production data volumes.** Median latency is misleading — the queries that time out are the ones users remember. A working system delivering sub-second results on real data beats a sophisticated parser that misses the latency requirement.

Practical Solution Pattern

Build a constrained intent parser that maps natural language to a known set of domain entities and structured queries. Search both primary records and AI-generated outputs through a unified interface. Enforce sub-second latency optimized against actual query patterns, and support mobile as a first-class search surface rather than a degraded desktop experience.

This works because it avoids the two failure modes that kill operational search projects: over-engineering the NLP layer to handle queries users never ask, and under-engineering the data layer so that accurately parsed queries still time out. The constrained parser stays reliable because its vocabulary is bounded by the schema. Speed to a working, measured deployment is what separates search systems that get adopted from those abandoned after a pilot — deep expertise paired with AI-augmented execution compresses the build cycle for systems where the NLP layer, the data architecture, and the performance requirements must all align from the start. If an operational search workflow is already defined and ready for build, **AI Workflow Integration** is the direct path to production.

References

1. Liu, M. and Xu, J. **NLI4DB: A Systematic Review of Natural Language Interfaces for Databases**. *arXiv*, 2025.
2. Liu, X. et al. **A Survey of Text-to-SQL in the Era of LLMs**. *arXiv*, 2024.
3. LANSA. **Natural Language Query: Simplifying Data Access**. *LANSA*, 2025.
4. Databricks. **Vector Search Performance Guide**. *Databricks Documentation*, 2025.
5. Microsoft. **Retrieval-Augmented Generation in Azure AI Search**. *Microsoft Learn*, 2026.
6. OpenSearch. **Revolutionizing Enterprise Search: OpenSearch Vector Engine**. *OpenSearch*, 2024.