CASE STUDY

Searching an Arabic dictionary by meaning

A RAG-based search that helps writers and translators find the word for a meaning they have in mind

In a small team (two engineers, a linguistics team and a designer), I built the AI search for an Arabic dictionary platform. It helps writers, translators and language researchers find the right word when they only know its meaning.



1 The problem

Writers, translators and researchers sometimes know exactly what they mean but not the word for it. A regular dictionary can’t help, because you need the word to look it up. A general chatbot can answer, but it may invent a word that doesn’t exist. Here, results always come from the dictionary’s own entries.

The linguistics team confirmed this gap, and language specialists said that solving it matters.

2 My role

I was the AI engineer. The first version sent every chapter and entry title to an LLM with each query and asked it to pick the closest match, which was too slow to be practical. I proposed rebuilding it as RAG and owned the decisions behind it: the query is turned into an embedding, a vector search finds the closest 100 titles, and the LLM picks the best match from that short list. The LLM returns only entry IDs in a fixed format, and the final text is fetched from the database, so every result shown is a real dictionary entry. Inputs are checked for length and malicious code.

I worked with a linguist to define what a good result looks like. I kept lists of search terms and their expected results as working notes, not a formal dataset, and used them to measure retrieval accuracy when comparing options. I also covered the full-stack work the product needed (search filters, the onboarding tour, the contact flow), joined product and UX discussions, put the product in front of people to hear their feedback, and proposed a smart-suggestions feature that shows the closest match instead of “no related result”.

3 Three decisions

- **An open-source Arabic embedding model instead of an API.** I tested three embedding models (two open-source, one API) on word pairs from the linguists, and chose an Arabic-focused open-source model for its accuracy. I gave up easy setup. Deploying it on SageMaker took an external embedding API out of the request path and gave a fixed running cost instead of per-request pricing, which matters as usage grows.
- **Embedding only entry titles and chapter names, not the explanations.** Titles carry the general meaning of a chapter, and I worried that embedding the longer explanations would blur its representation. Results were already good, so I gave up further testing before launch. We launched on time, and I planned to test the other ideas afterwards.
- **Query processing in the backend, with SageMaker used only to host the embedding model.** I gave up having everything in one managed service. I used the smallest instance that worked to keep hosting cost low.

Result

The product launched with real users and passed 1,000 searches in its first week. In my testing, response time dropped from 9+ seconds to typically under 3, often faster, depending on the API service's response. The LLM now receives about 100 lines per query instead of about 3,500.

1,000+	9+ s to under 3 s	3,500 to 100
searches in the first week (approx.)	typical response time (informal testing)	lines sent to the LLM per query (approx.)

4 What I'd test next

The search sends the LLM only the closest 100 results, so if the right chapter isn't in that list, the LLM can't choose it. I compared options on retrieval accuracy before launch, but I haven't measured how often the right chapter falls outside the top 100 in real searches.

I'd run my lists of search terms and expected results through the vector search alone, without the LLM, and record where the correct chapter lands: within the top 100, 200 or 300. If most land within 100, the weak point is the LLM's choice, and I'd revisit the prompt. If many land further down, I'd raise the number of results or reconsider which text is embedded. The number I'd watch is the share of test searches whose correct chapter appears within each cutoff.

5 Honest label

Real product with real users. There was no formal benchmark dataset: I used working lists of search terms and expected results from the linguists, and search quality was judged by the linguistics team's review. Usage figures are approximate. Response times were measured informally, not benchmarked. Cost advantages are expected, not measured.