Faculte des sciences de Bizert | 2025/2026

Project 2: NLP & Semantic Matching

Presented by Tiouajni Sirine

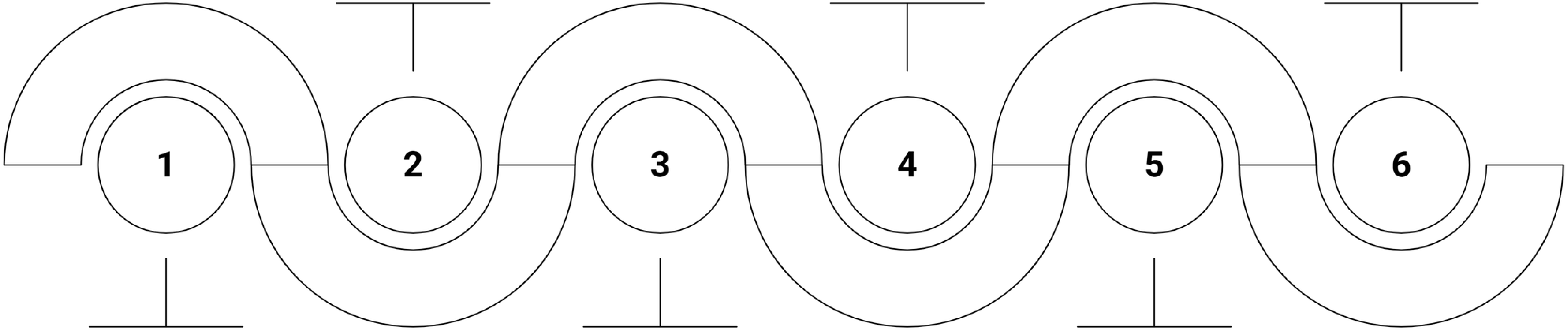
Problem

Imagine an industrial process stopping just because of a small typo. A worker types "Bosch indstrl screwdriver" instead of the exact catalog name, and the system finds nothing. In real life,descriptions are messy: using shortcuts, making spelling mistakes, and describing things in our own way.

So how can machines understand what people mean, not just what they type?

This project answers that by going beyond keywords and focusing on real semantic understanding.

Methodology



Data Management

- Data is versioned using DVC to ensure reproducibility without bloating the Git repository.

Preprocessing

- Raw catalog data is normalized by converting to lowercase, removing special characters/accents, and collapsing extra whitespace.

Semantic Core

- Text is converted into 384-dimensional dense vectors using the all-MiniLM-L6-v2 model, chosen for its speed and compact size.

Indexing

- Fast similarity search is powered by FAISS (IndexFlatIP), which calculates inner products for cosine similarity.

The Hybrid Strategy

- To maximize accuracy, the system combines semantic retrieval with a Cross-Encoder reranker for fine-grained relevance and a BM25 lexical fallback for cases where semantic scores are too low.

Deployment

- The solution is served via a FastAPI endpoint and fully containerized using Docker for consistent production deployment.

Data Management & Versioning

I worked with a fairly large dataset, so I used DVC to manage it efficiently.

Data Preprocessing

- Before applying NLP, I cleaned both the equipment catalog and user inputs.
- I normalized text by fixing casing, special characters, and extra spaces.

→ This step reduces noise caused by typos and inconsistent writing.

Embedding Generation

- I used a pre-trained sentence embedding model to understand meaning.
- Each equipment name is converted into a numeric vector.

→ This allows the system to compare items based on semantic similarity, not keywords.

Vector Indexing with FAISS

- I indexed all vectors using FAISS for fast similarity search.

→ This makes retrieval efficient and scalable.

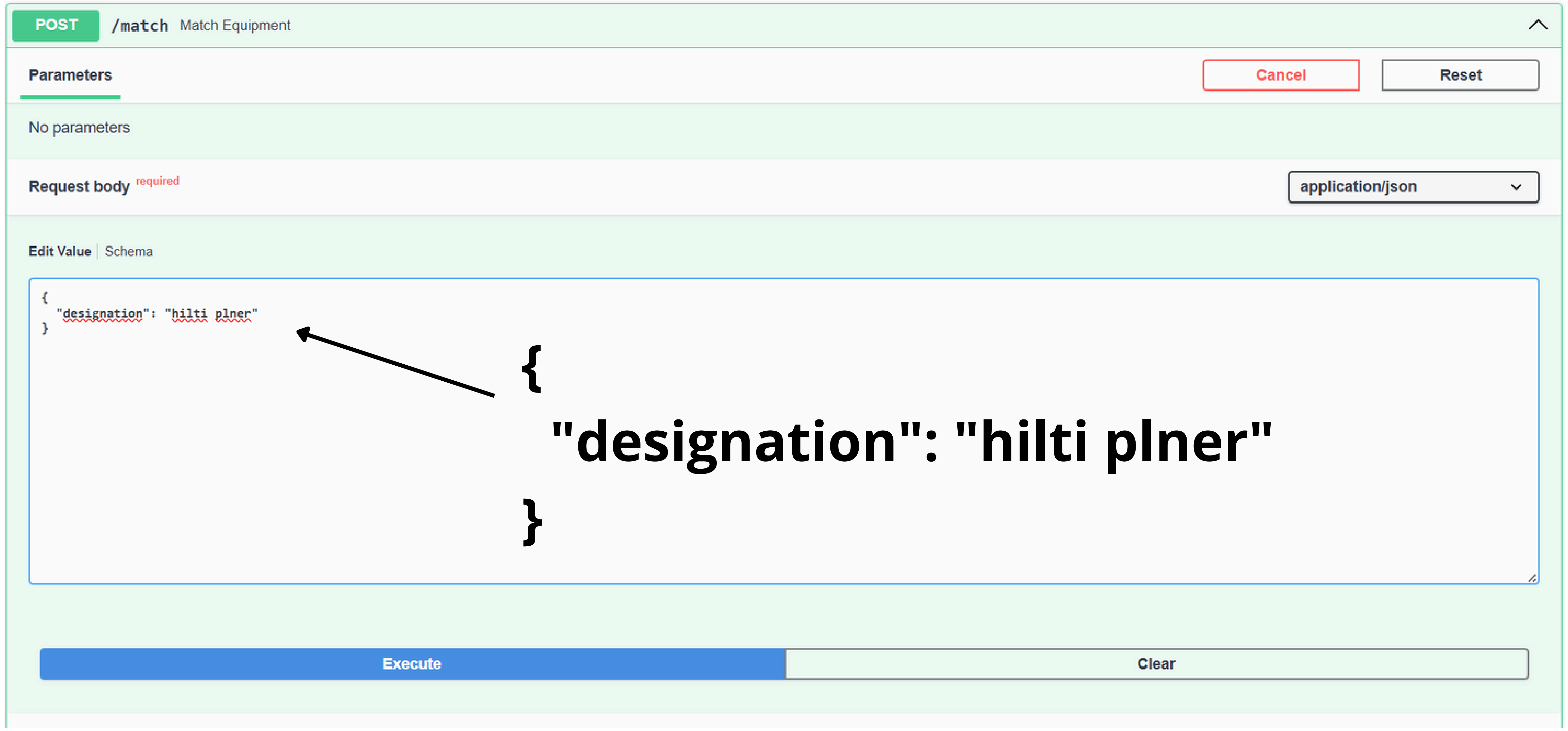
→ The index can be reused without recomputing embeddings.

Hybrid Search & Retrieval

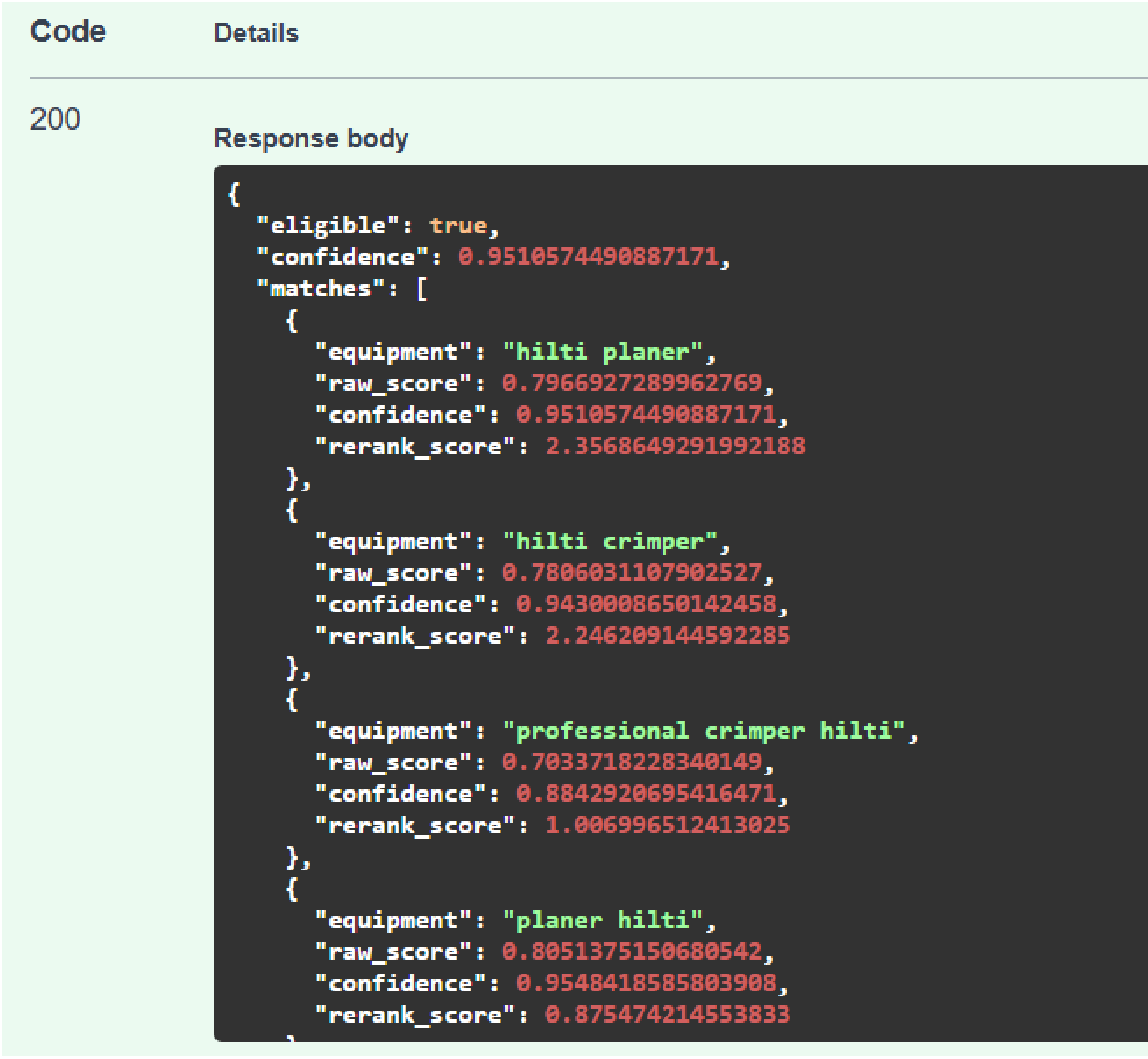
- I combined semantic search with a lexical fallback.
- This helps handle rare words, brands, and spelling mistakes.
- The system stays reliable even with messy real-world input.

Evaluation & Results

FastAPI



In http://localhost:8000/docs



evaluation.py

```
[
  {
    "query": "black+decker bfs",
    "expected": "Black+Decker Face Shield"
  },
  {
    "query": "makita makita clamp meter",
    "expected": "Makita Clamp Meter"
  },
  {
    "query": "hilti professional jigsaw",
    "expected": "Hilti Professional Jigsaw"
  },
  {
    "query": "dewalt dtm",
    "expected": "DeWalt Tape Measure"
  },
  {
    "query": "stanley",
    "expected": "Compact Protractor Stanley"
  },
```

It loads 250 test queries from test_queries_clean.json.
For each query, it searches using SemanticSearch, measures latency, and checks if the expected equipment is in the top results.
Metrics are averaged across all queries.

evaluation.py

```
(.venv) PS C:\Users\Dell\Desktop\DvopsMlops> python evaluation.py
Evaluated 250 queries
Avg Latency: 0.0563s
Recall@1: 0.284
Recall@5: 0.56
MRR: 0.39759999999999995
(.venv) PS C:\Users\Dell\Desktop\DvopsMlops>
```

Recall@1: 28% → correct result ranked first in 1 out of 3 cases
Recall@5: 56% → correct result appears in the top 5 results
MRR: 0.40 → correct result usually ranked around position 2–3
Latency: ~0.06s → real-time response

Faculte des sciences de Bizert | 2025/2026

application testing by running the Docker image