REDISCOVERING APOLLO 11

Using Java, Redis, and Vector Search

Raphael De Lio

Redis

CONTEXT

FLIGHT PLAN

OBJECTIVE

”That’s one small step for a human, one big jump for humanity.”

IS SIMILAR TO

”That’s one small step for man, one giant leap for Mankind.”

VECTOR SIMILARITY SEARCH

VECTOR SIMILARITY SEARCH

How does it work?

”That’s one small step for a human, one big jump for humanity.”

[0.56, 0.76, 0.80, 0.54, 0.99, -0.87 ...]

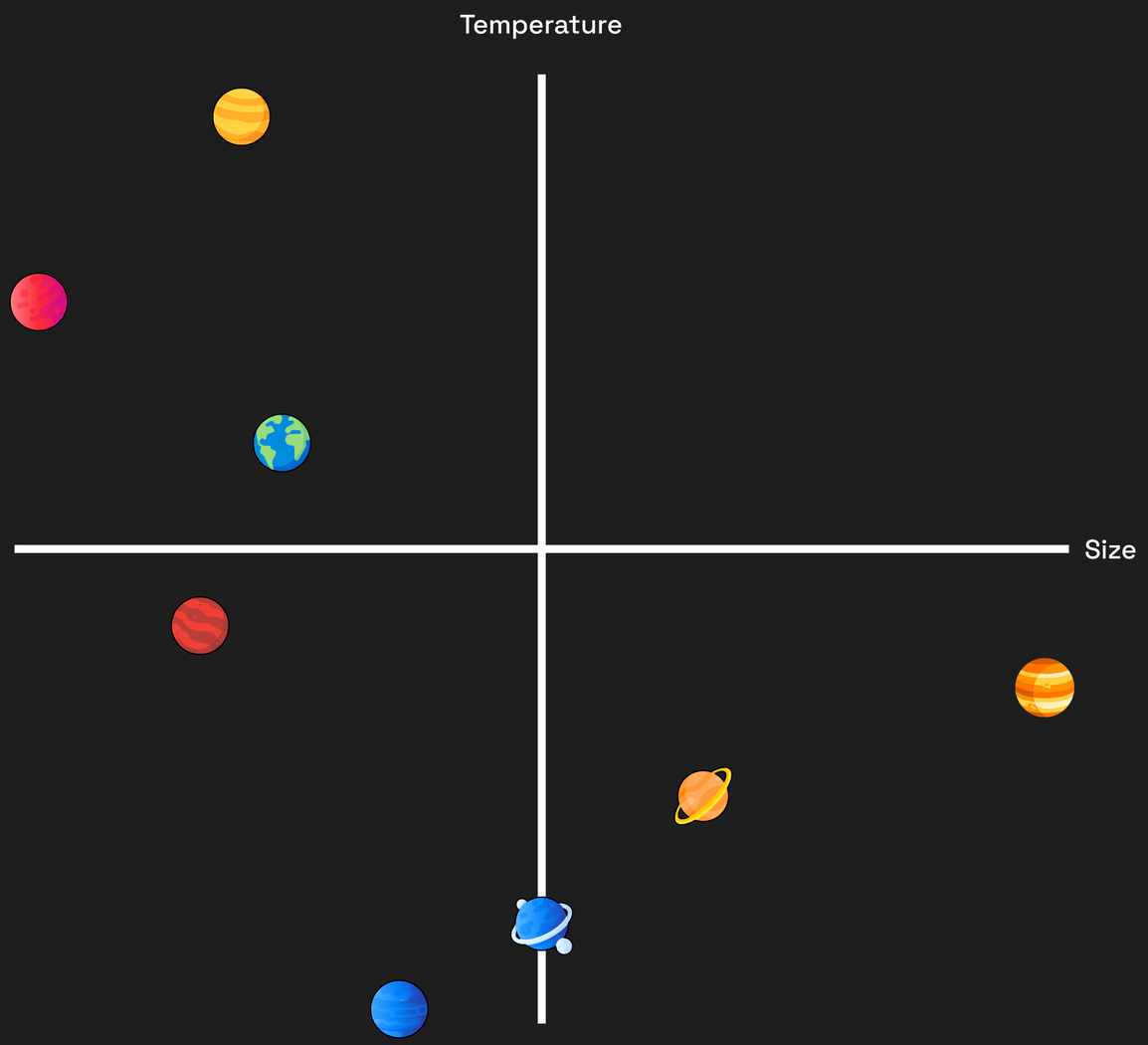
”That’s one small step for man, one giant leap for Mankind.”

[0.58, 0.75, 0.75, 0.63, 0.9, -0.7 ...]

VECTOR SIMILARITY SEARCH

How does it work?

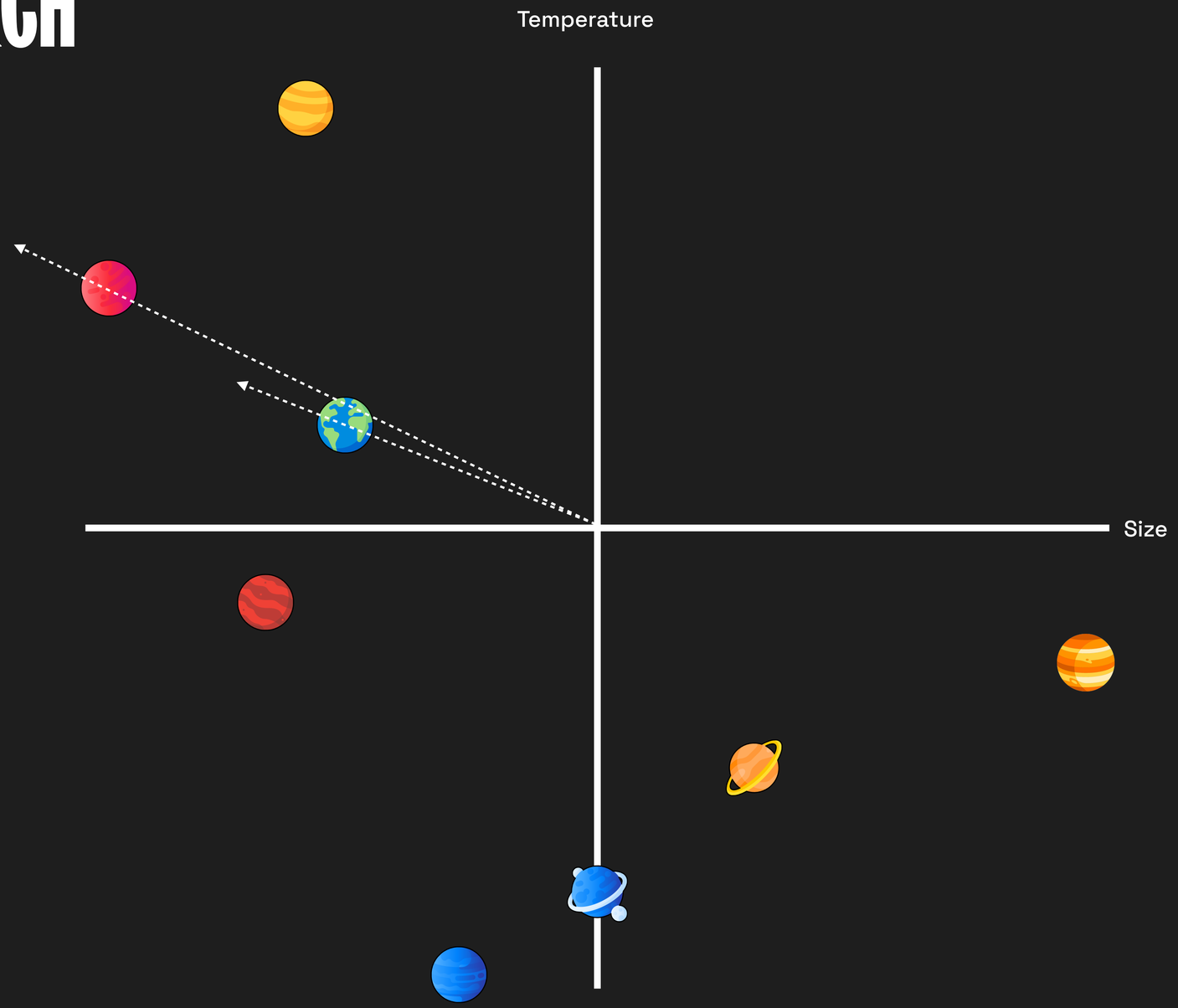
Planet	Size	Temp	Vector
Mercury	0,33	167	(-1, 0.5)
Venus	4,87	464	(-0.6, 1)
Earth	5,97	15	(-0.5, 0.2)
Mars	0,642	-65	(-0.7, -0.1)
Jupiter	1888	-110	(1.0, -0.3)
Saturn	568	-140	(0.3, -0.5)
Uranus	86.8	-195	(0, -0.9)
Neptune	102	-200	(-0.2, -1)



VECTOR SIMILARITY SEARCH

How do we search?

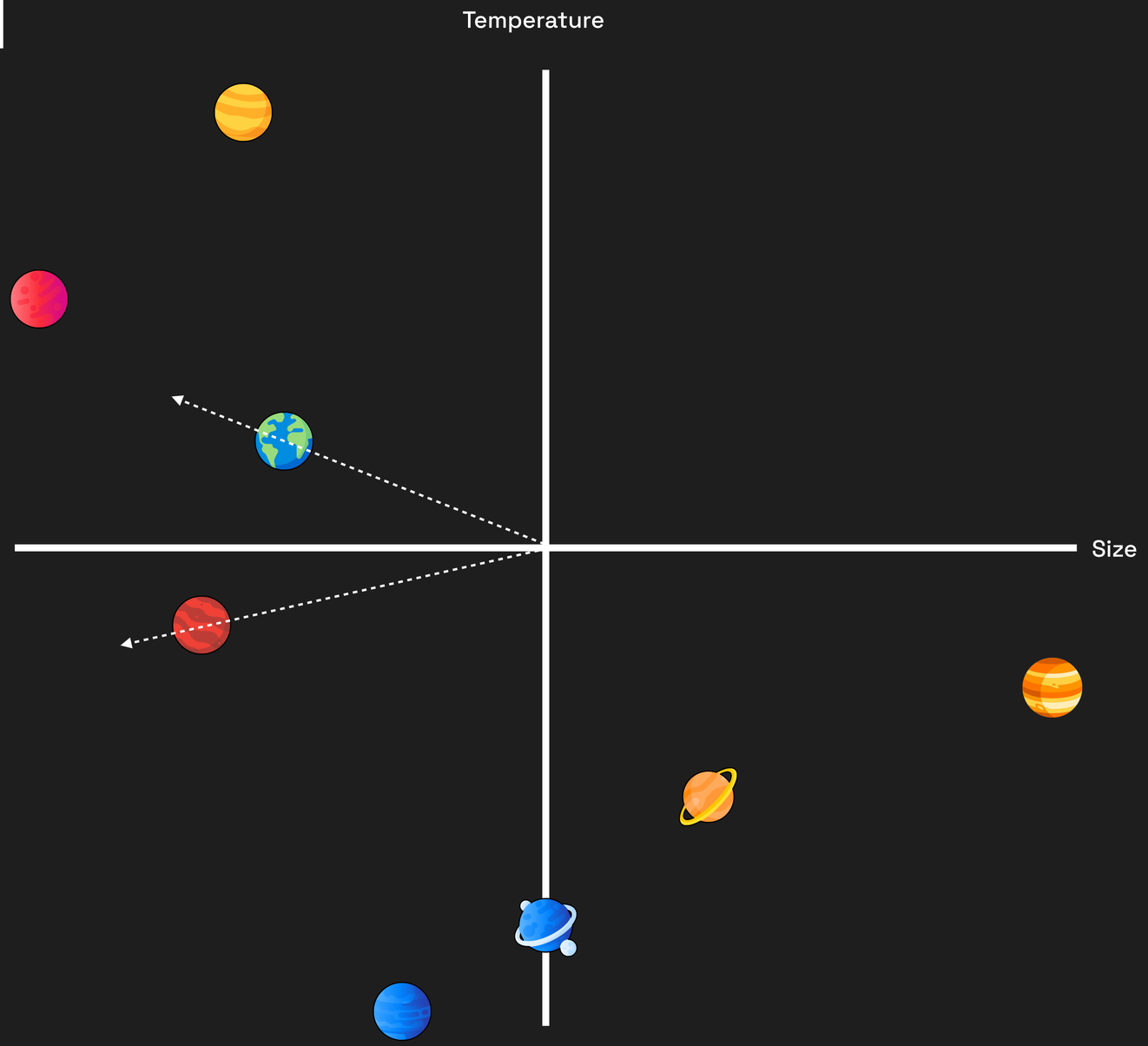
Cosine Similarity
Angle Based



VECTOR SIMILARITY SEARCH

How do we search?

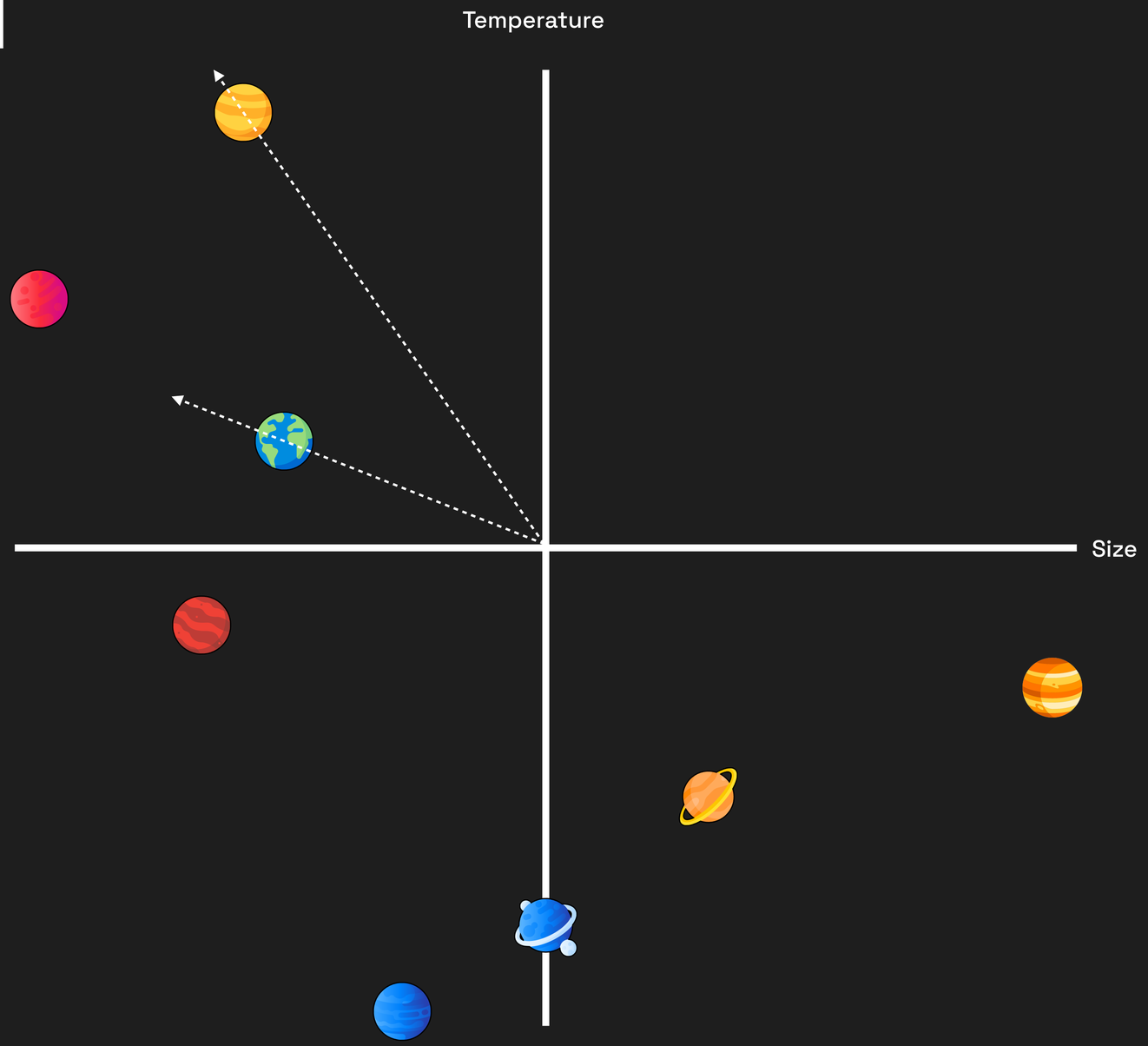
Cosine Similarity
Angle Based



VECTOR SIMILARITY SEARCH

How do we search?

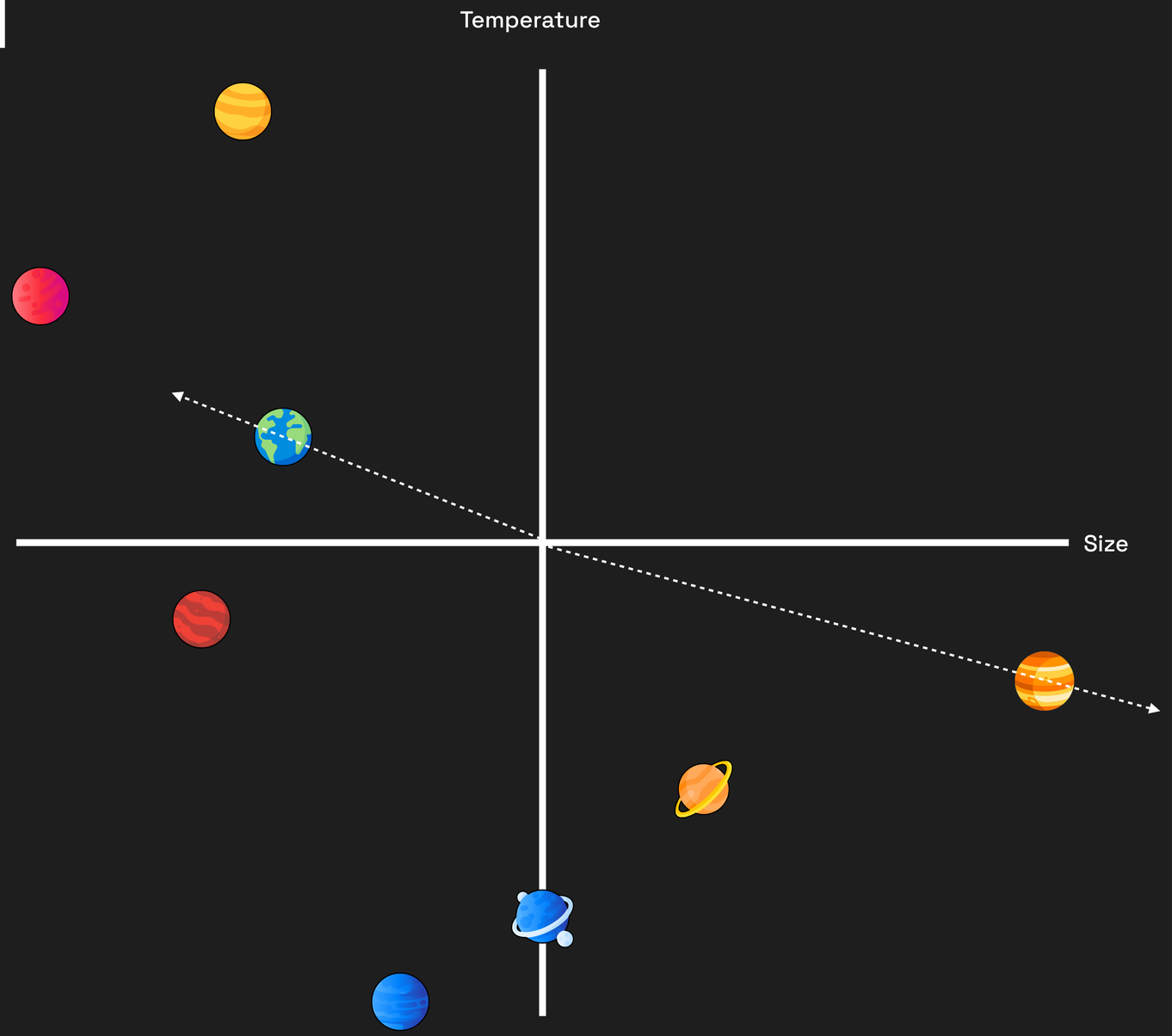
Cosine Similarity
Angle Based



VECTOR SIMILARITY SEARCH

How do we search?

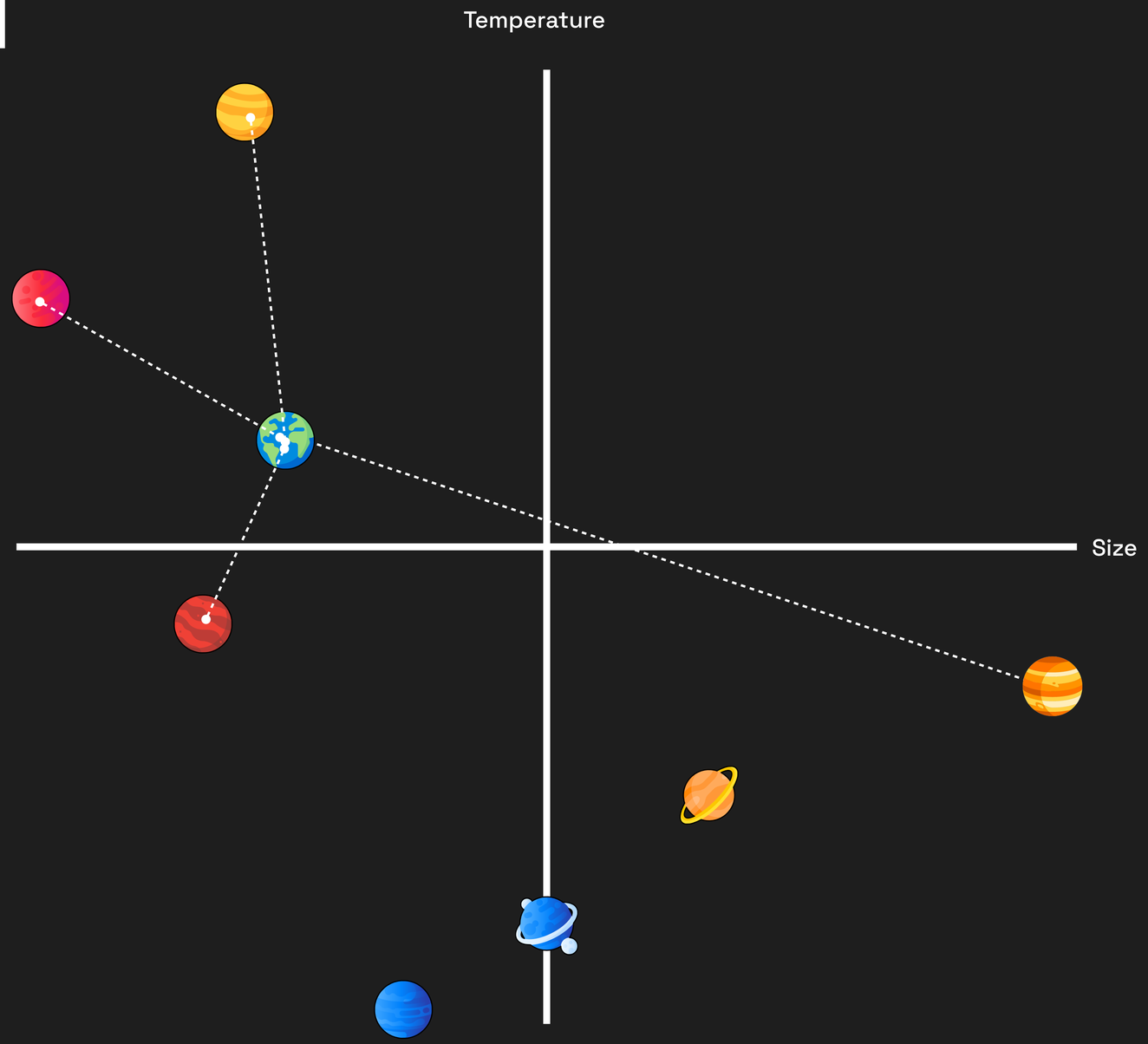
Cosine Similarity
Angle Based



VECTOR SIMILARITY SEARCH

How do we search?

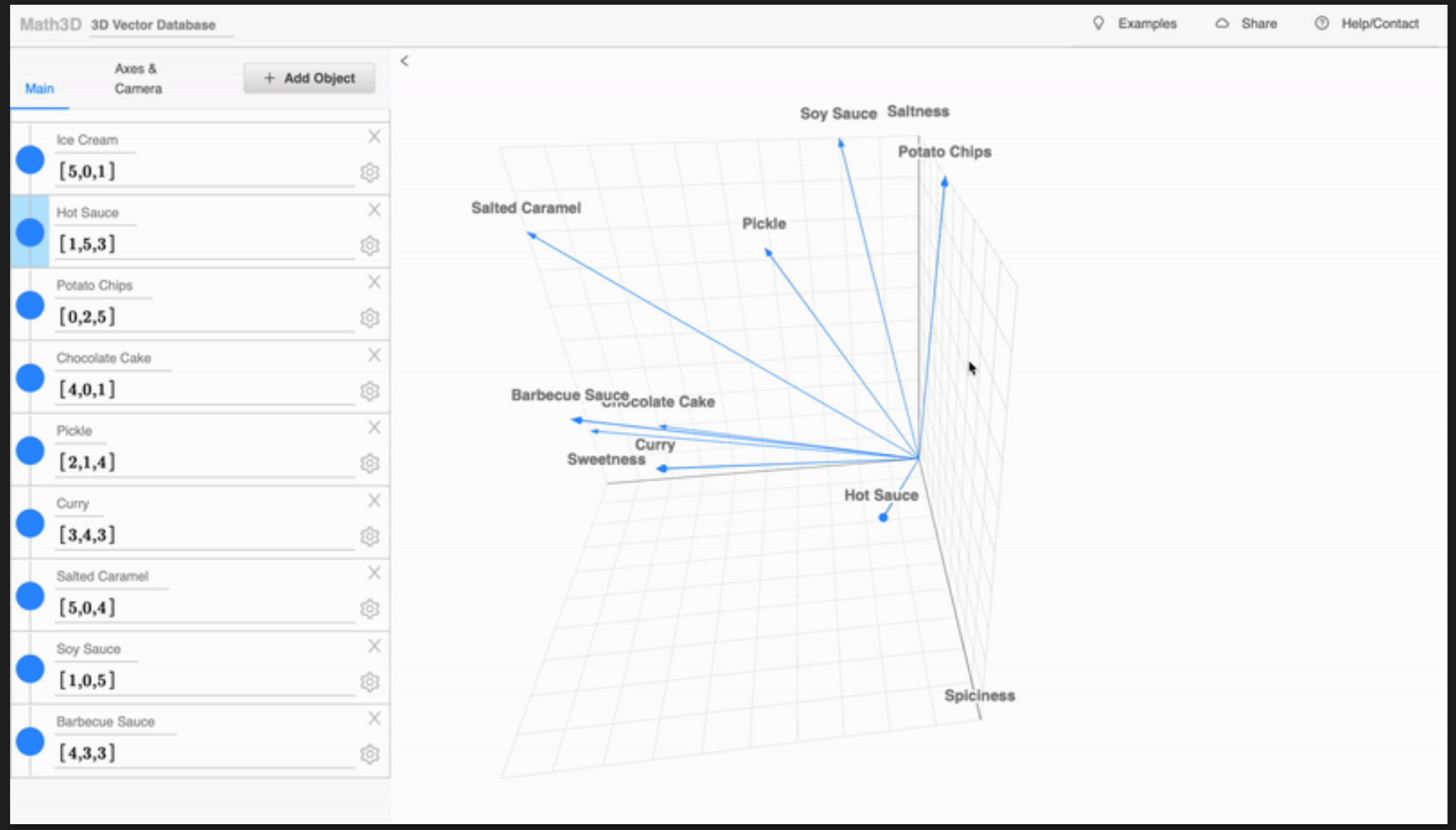
Euclidean Distance
Magnitude Based

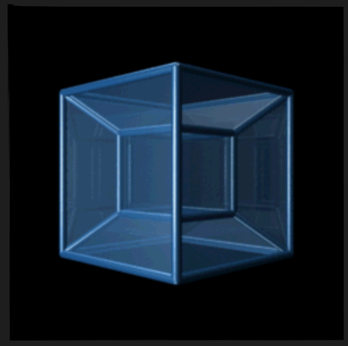


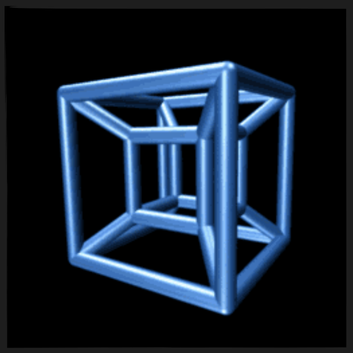
VECTOR SIMILARITY SEARCH

How many dimensions?

*heads up de como escolher o embedding model



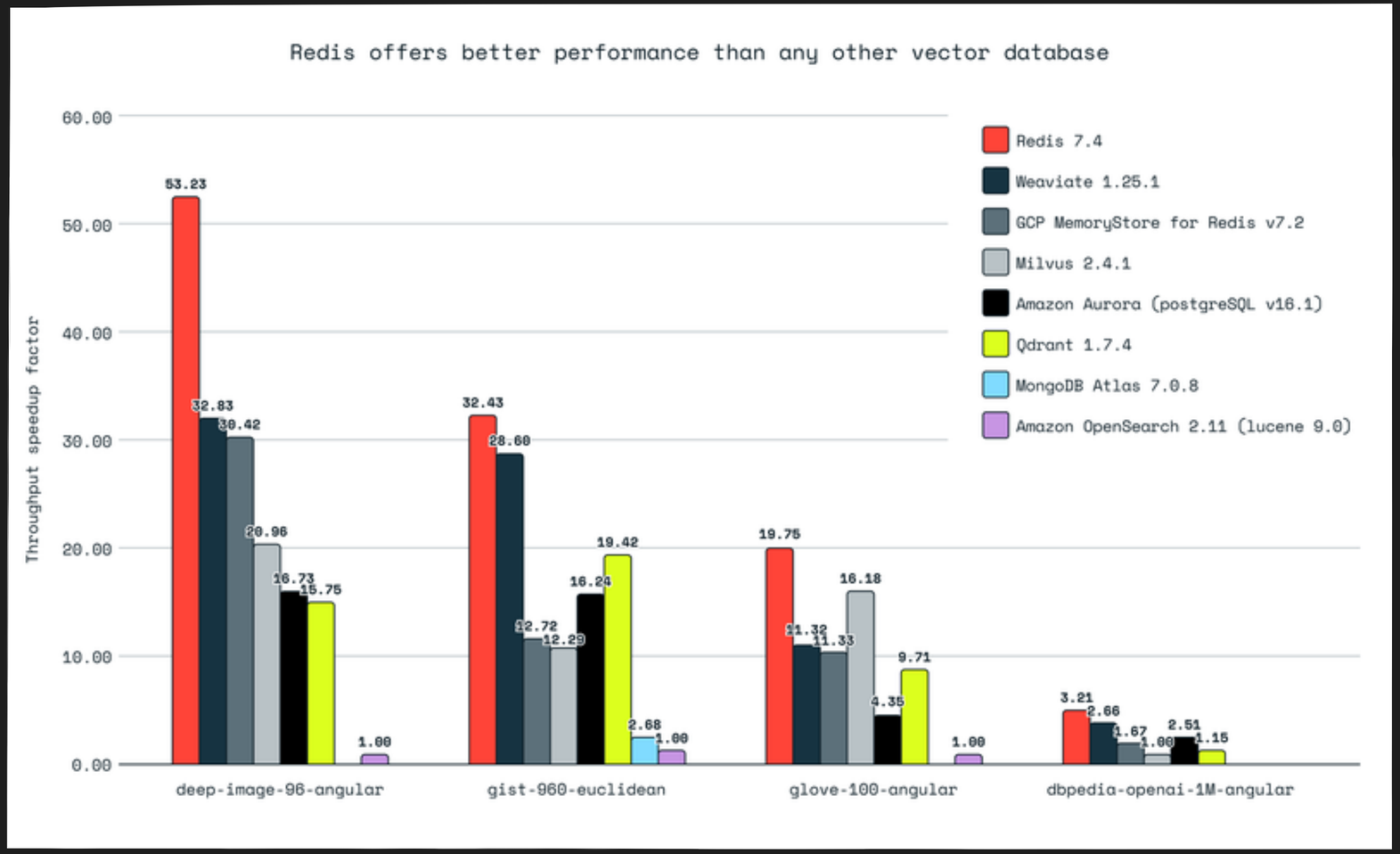




VECTOR SIMILARITY SEARCH

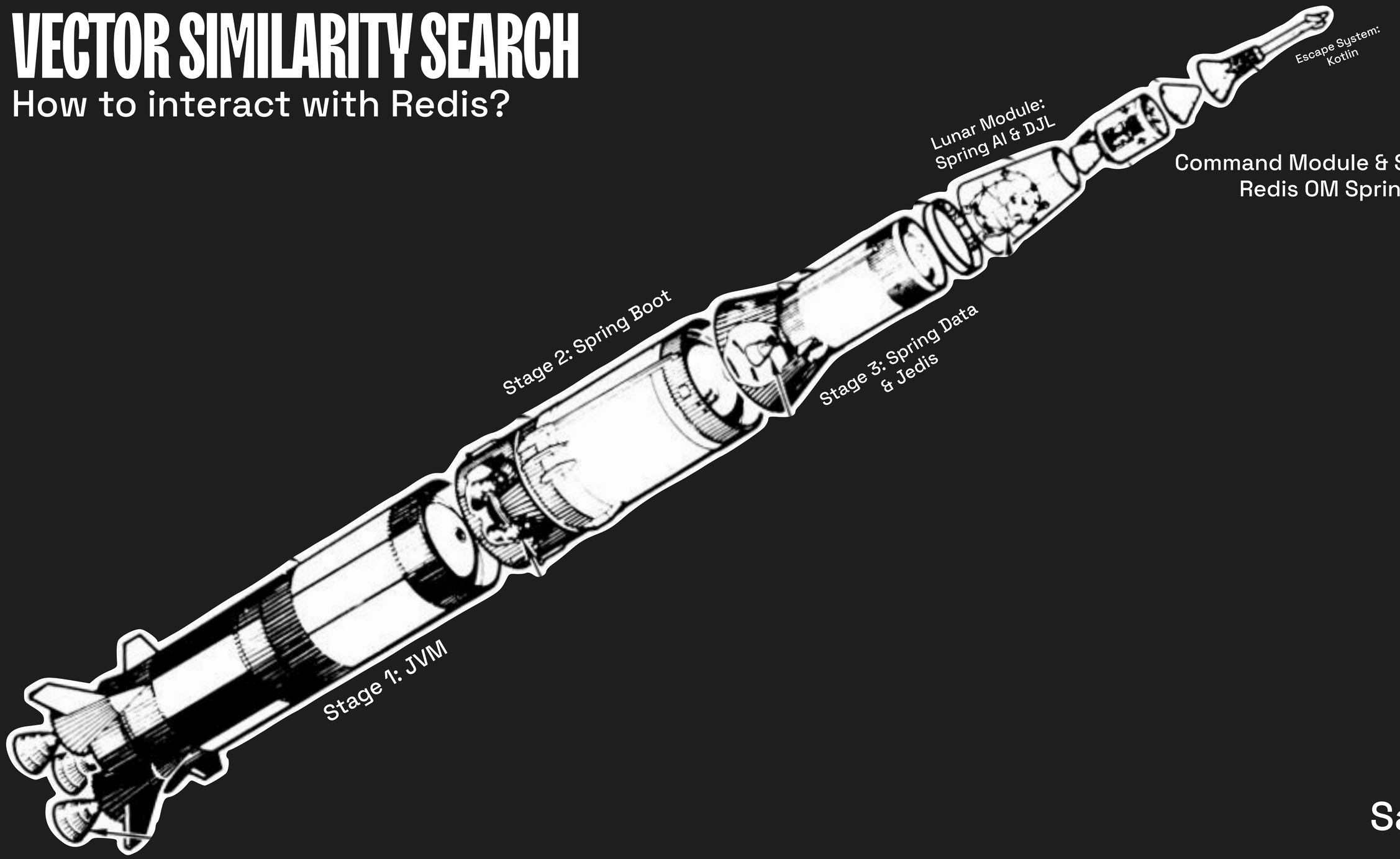
Where to store our vectors?

Redis



VECTOR SIMILARITY SEARCH

How to interact with Redis?



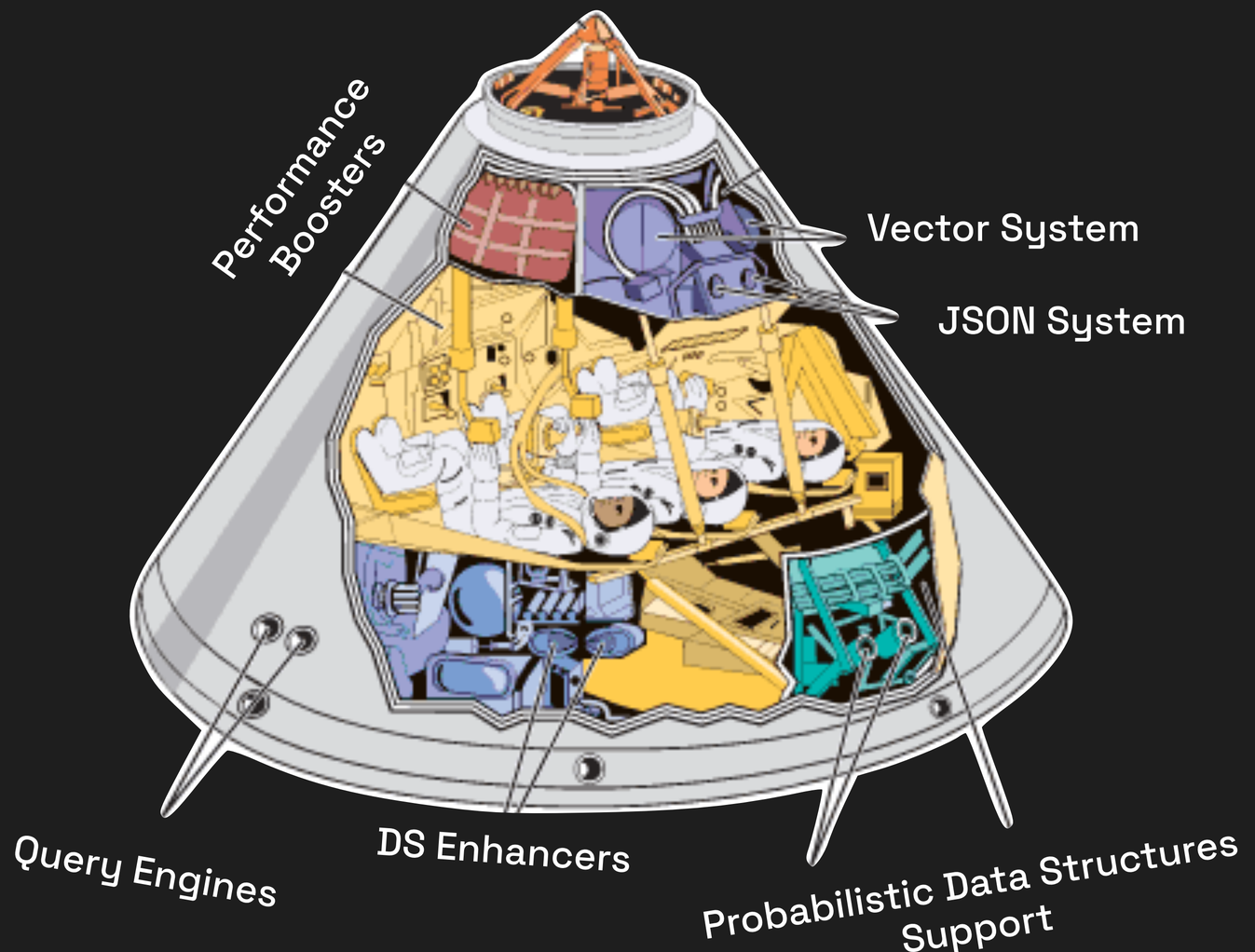
Saturn V

VECTOR SIMILARITY SEARCH

How to interact with Redis?

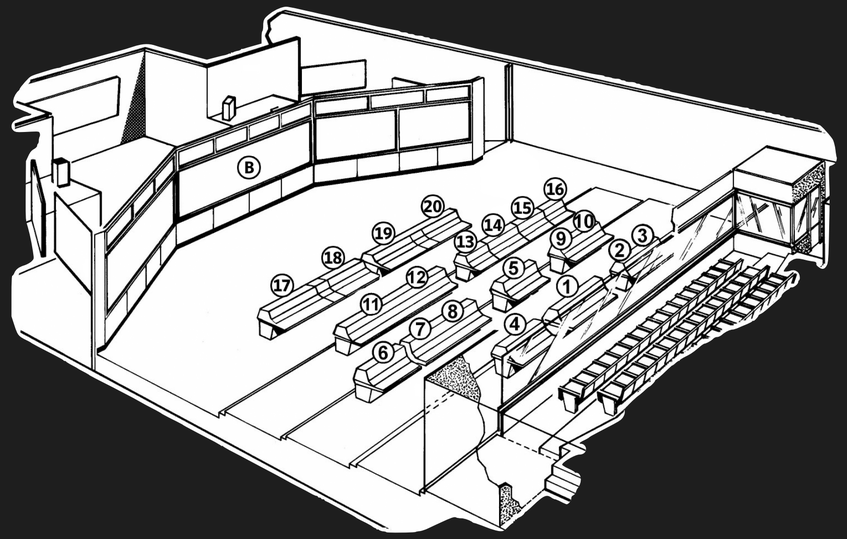
The Command Module

Redis OM Spring

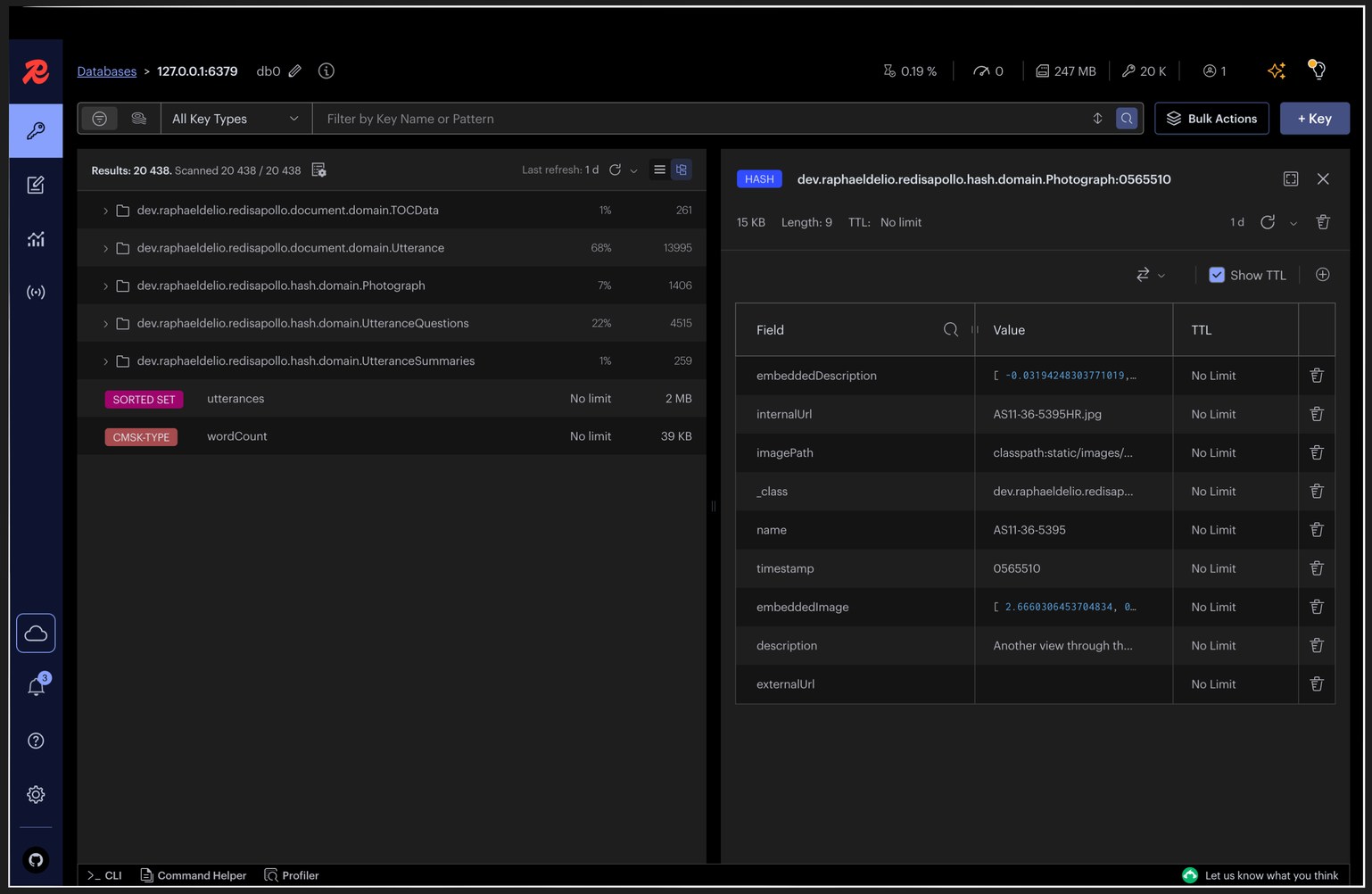


VECTOR SIMILARITY SEARCH

How to interact with Redis?



Mission Control



TAKING OFF

- Collect our data
- Load our data into Redis
- Query our data

COLLECTING THE DATA


```
{
  [
    "1092339",
    "Armstrong",
    "I'm at the foot of the ladder. The LM footpads are only depressed in the surface about 1 or 2 inches, although the surface appears to be very, very fine grained, as you get close to it. It's almost like a powder. - ground mass is very fine",
    "T"
  ],
  [
    "1092414",
    "Armstrong",
    "Okay. I'm going to step off the LM now.",
    "T"
  ],
  [
    "1092426",
    "Armstrong",
    "That's one small step for man, one giant leap for Mankind.",
    "T"
  ]
}
```

LOADING THE DATA

```java
@Data
@RedisHash
public class Utterance {

    @Id
    private String timestamp;

    @Indexed
    @NonNull
    private String speaker;

    @Indexed
    @NonNull
    private String text;

    @Indexed
    @NonNull
    private String speakerId;
}
```

Annotation for defining a HASH object

Annotation for allowing efficient querying based on that field's value in Redis.

Creating the repository

```java
public interface UtteranceRepository extends RedisEnhancedRepository<Utterance, String>
```

VECTORIZING THE DATA

```
1     @NonNull
2     @Vectorize(
3             destination = "embeddedText",
4             embeddingType = EmbeddingType.SENTENCE
5     )
6     private String text;
7
8     @Indexed(
9             schemaFieldType = SchemaFieldType.VECTOR,
10            algorithm = VectorField.VectorAlgorithm.HNSW,
11            type = VectorType.FLOAT32,
12            dimension = 768,
13            distanceMetric = DistanceMetric.COSINE,
14            initialCapacity = 10
15    )
16    private byte[] embeddedText;
```

Data to be vectorized

Field where the vectorize data will be stored in

VECTORIZING THE DATA

```
1     @NonNull
2     @Vectorize(
3             destination = "embeddedText",
4             embeddingType = EmbeddingType.SENTENCE,
5             provider = EmbeddingProvider.OPENAI,
6             openAiEmbeddingModel = OpenAiApi.EmbeddingModel.TEXT_EMBEDDING_3_LARGE
7     )
8     private String text;
9
10    @Indexed(
11            schemaFieldType = SchemaFieldType.VECTOR,
12            algorithm = VectorField.VectorAlgorithm.HNSW,
13            type = VectorType.FLOAT32,
14            dimension = 3072,
15            distanceMetric = DistanceMetric.COSINE,
16            initialCapacity = 10
17    )
18    private byte[] embeddedText;
```

Different embedding model

Number of dimensions must match the number of dimensions of the model

VECTORIZING THE DATA

```
1     @NonNull
2     @Vectorize(
3             destination = "embeddedImage",
4             embeddingType = EmbeddingType.IMAGE
5     )
6     private String imagePath;
7
8     @Indexed(
9             schemaFieldType = SchemaFieldType.VECTOR,
10            algorithm = VectorField.VectorAlgorithm.HNSW,
11            type = VectorType.FLOAT32,
12            dimension = 512,
13            distanceMetric = DistanceMetric.COSINE,
14            initialCapacity = 10
15    )
16    private byte[] embeddedImage;
```

QUERY THE DATA

Vectorizing the query

```
1 byte[] embedding = embedder.getTextEmbeddingsAsBytes(List.of(queryText), Utterance$.TEXT).get(0);
2
3 SearchStream<Utterance> stream = entityStream.of(Utterance.class);
4 List<Pair<Utterance, Double>> textsAndScores = stream
5         .filter(Utterance$.EMBEDDED_TEXT.knn(3, embedding))
6         .sorted(Utterance$._EMBEDDED_TEXT_SCORE)
7         .map(Fields.of(Utterance$._THIS, Utterance$._EMBEDDED_TEXT_SCORE))
8         .collect(Collectors.toList());
```

Searching for the 3 nearest neighbors

This is going to generate the command to search and sort on Redis.
This is done efficiently by the Redis Query Engine

DEMO

HOUSTON, WE HAVE A PROBLEM

GRANULARITY PROBLEM

- Short text: “Apollo” → Doesn’t tell if it’s about a Greek god, a space mission, or a music album.
- Long text: A full transcript of a mission → May contain too much irrelevant information for a specific query.

*Explicar como isso afeta a utilização de vector search

TABLE OF CONTENTS

SUMMARIZATION

```java
public void summarize(boolean overwrite) {
    logger.info("Summarizing TOC entries");
    List<TOCData> tocDataList = tocDataRepository.findAll();

    for (TOCData tocData : tocDataList) {
            ChatResponse response = chatModel.call(
                    new Prompt(List.of(
                            new SystemMessage("""
                                    You are a helpful assistant who summarizes utterances of the Apollo 11 mission.
                                    Make these summaries very dense with all curiosities included.
                                    Limit the summary to 512 words.
                                    """),
                            new UserMessage(tocData.getConcatenatedUtterances())
                    ))
            );

            tocData.setSummary(response.getResult().getOutput().getContent());
            tocDataRepository.save(tocData);
            logger.info("Summarized TOC entry: {}", tocData.getStartDate());
    }

    logger.info("Summarized TOC entries");
}

```

EXTRACTION

```
logger.info("Generating questions for TOC entries");
List<TOCData> tocDataList = tocDataRepository.findAll();
for (TOCData tocData : tocDataList) {
        ChatResponse response = chatModel.call(
                new Prompt(List.of(
                        new SystemMessage("""
                                You are a helpful assistant that is helping me predict which questions can be asked by
                                people who are trying to rediscover the Apollo 11 mission data. You will be given a number of
                                utterances and you will predict the questions that can be asked by people who are trying to
                                rediscover the Apollo 11 mission data. You will ONLY return the questions separate by breaklines,
                                and nothing more. You will NEVER return more than 512 words.
                                """),
                        new UserMessage(tocData.getConcatenatedUtterances())
                ))
        );

        var questions = Arrays.stream(response.getResult().getOutput().getContent().split("\n"))
                .filter(q ->!q.isBlank()).toList();
        tocData.setQuestions(questions);
        tocDataRepository.save(tocData);
        logger.info("Generated questions for TOC entry: {}", tocData.getStartDate());
}
logger.info("Generated questions for TOC entries");
```

QUERYING

Searching on the questions instead of texts

```
SearchStream<UtteranceQuestions> stream = entityStream.of(UtteranceQuestions.class);
List<Pair<UtteranceQuestions, Double>> questionsAndScores = stream
        .filter(UtteranceQuestions$.EMBEDDED_QUESTION.knn(3, embeddingToBeQueried))
        .sorted(UtteranceQuestions$._EMBEDDED_QUESTION_SCORE)
        .map(Fields.of(UtteranceQuestions$._THIS, UtteranceQuestions$._EMBEDDED_QUESTION_SCORE))
        .collect(Collectors.toList());
```

RETRIEVAL AUGMENTED GENERATION

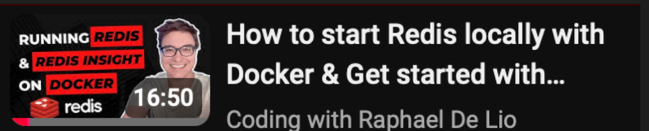
- Feed the LLM with the related utterances
- Ask it to answer our question based only on the information we provide

```java
@PostMapping("/search-by-question")
public Map<String, Object> searchByQuestion(@RequestBody Map<String, String> requestBody) {
    String queryText = requestBody.get("query");
    logger.info("Received question: {}", queryText);

    byte[] embedding = embedder.getTextEmbeddingsAsBytes(List.of(queryText), UtteranceQuestions$.QUESTION).get(0);
    SearchStream<UtteranceQuestions> stream = entityStream.of(UtteranceQuestions.class);
    List<Pair<UtteranceQuestions, Double>> questionsAndScores = stream
            .filter(UtteranceQuestions$.EMBEDDED_QUESTION.knn(3, embedding))
            .sorted(UtteranceQuestions$._EMBEDDED_QUESTION_SCORE)
            .map(Fields.of(UtteranceQuestions$._THIS, UtteranceQuestions$._EMBEDDED_QUESTION_SCORE))
            .collect(Collectors.toList());

    List<Map<String, String>> matchedQuestions = questionsAndScores.stream()
            .map(pair -> {
                UtteranceQuestions question = pair.getFirst();
                return Map.of(
                        "question", question.getQuestion(),
                        "utterances", question.getUtterancesConcatenated(),
                        "score", pair.getSecond().toString()
                );
            })
            .collect(Collectors.toList());
```

```
    ChatResponse response = chatModel.call(
            new Prompt(List.of(
                    new SystemMessage("""
                You are an expert assistant specializing in the Apollo missions. Your goal is to provide accurate,
                detailed, and concise answers to user inquiries by utilizing the provided Apollo mission data.
                Rely solely on the information given below and avoid introducing external information.
                """),
                    new SystemMessage("Apollo mission data: " + questionsAndScores.stream()
                            .map(it -> it.getFirst().getUtterancesConcatenated())
                            .collect(Collectors.joining("\n"))),
                    new UserMessage("User question: " + queryText)
            ))
    );

    logger.info("AI response: {}", response.getResult().getOutput().getContent());
    return Map.of(
            "query", queryText,
            "answer", response.getResult().getOutput().getContent(),
            "matchedQuestions", matchedQuestions
    );
}
```

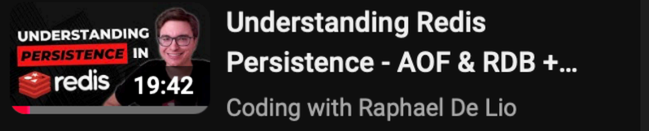
DEMO

TAKEAWAYS



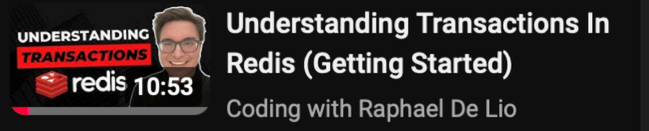
How to start Redis locally with Docker & Get started with...

Coding with Raphael De Lio



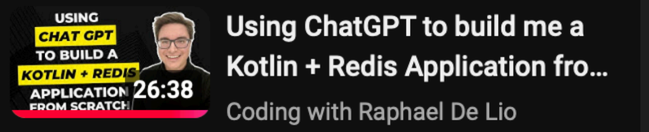
Understanding Redis Persistence - AOF & RDB +...

Coding with Raphael De Lio



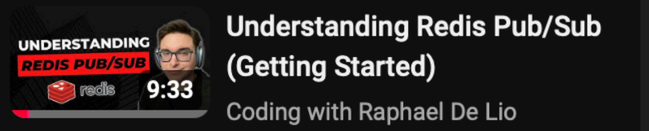
Understanding Transactions In Redis (Getting Started)

Coding with Raphael De Lio



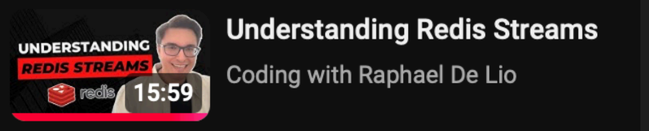
Using ChatGPT to build me a Kotlin + Redis Application fro...

Coding with Raphael De Lio



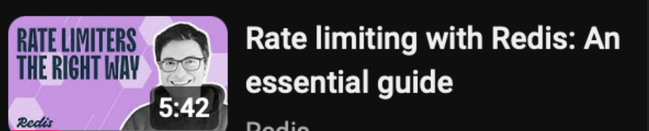
Understanding Redis Pub/Sub (Getting Started)

Coding with Raphael De Lio



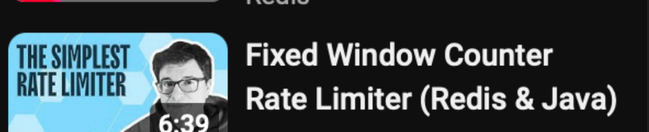
Understanding Redis Streams

Coding with Raphael De Lio



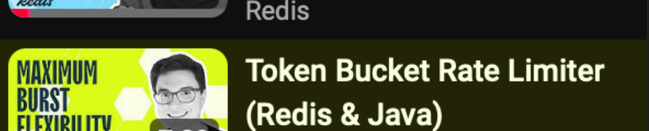
Rate limiting with Redis: An essential guide

Redis



Fixed Window Counter Rate Limiter (Redis & Java)

Redis



Token Bucket Rate Limiter (Redis & Java)

Redis

RAPHAEL DE LIO

DEVELOPER ADVOCATE

Redis