

# 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

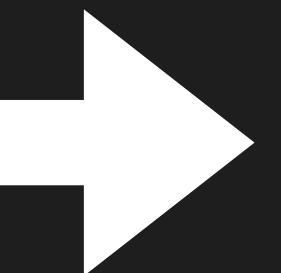
“That’s one small step for man, one giant leap for Mankind.”

# VECTOR SIMILARITY SEARCH

# VECTOR SIMILARITY SEARCH

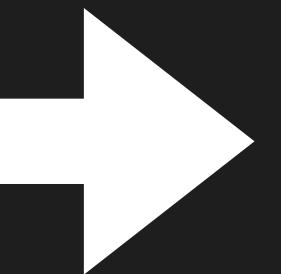
How does it work?

**"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."**

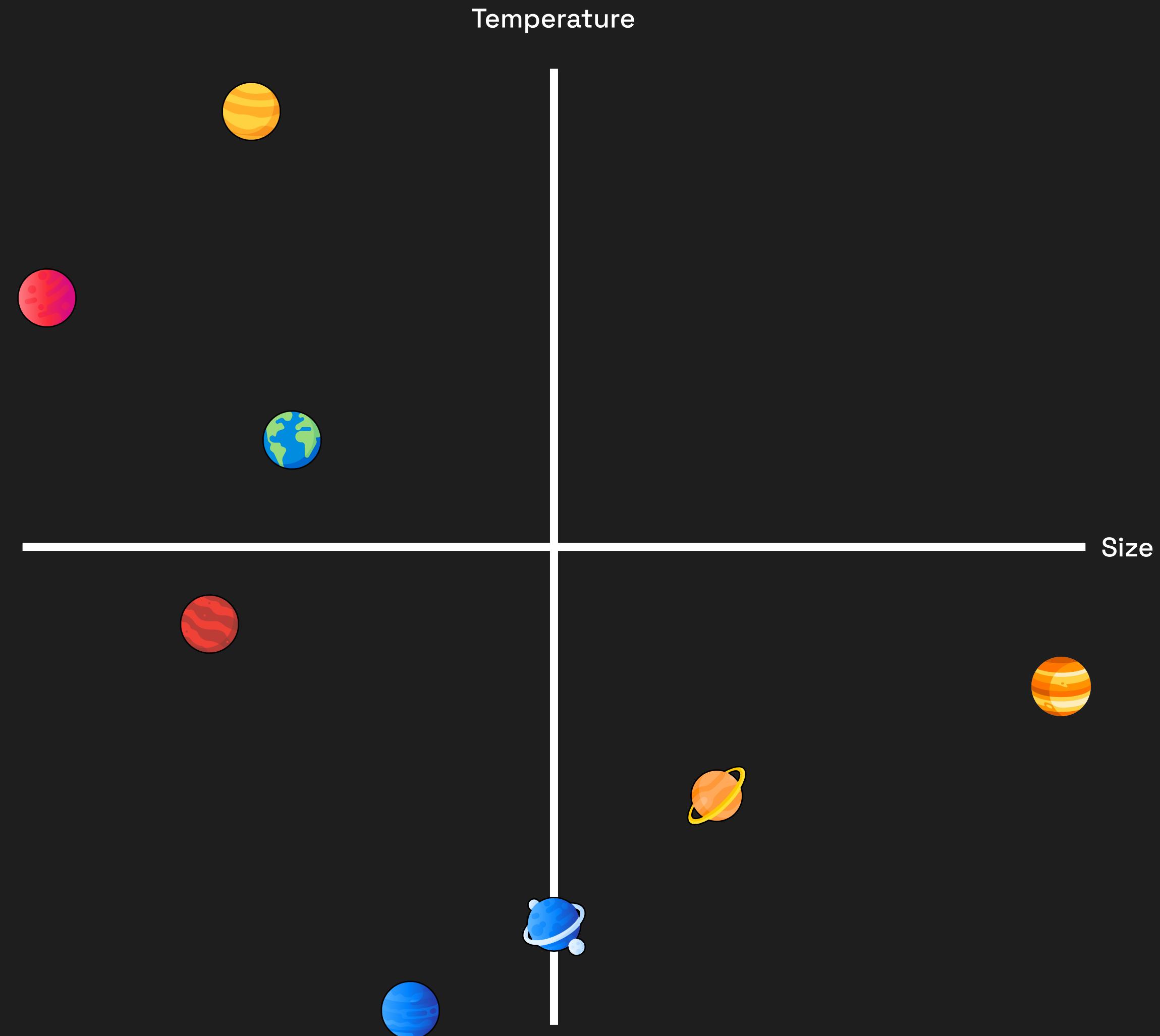


[0.58, 0.75, 0.75, 0.63, 0.9, -0.7 ... ]

# VECTOR SIMILARITY SEARCH

How does it work?

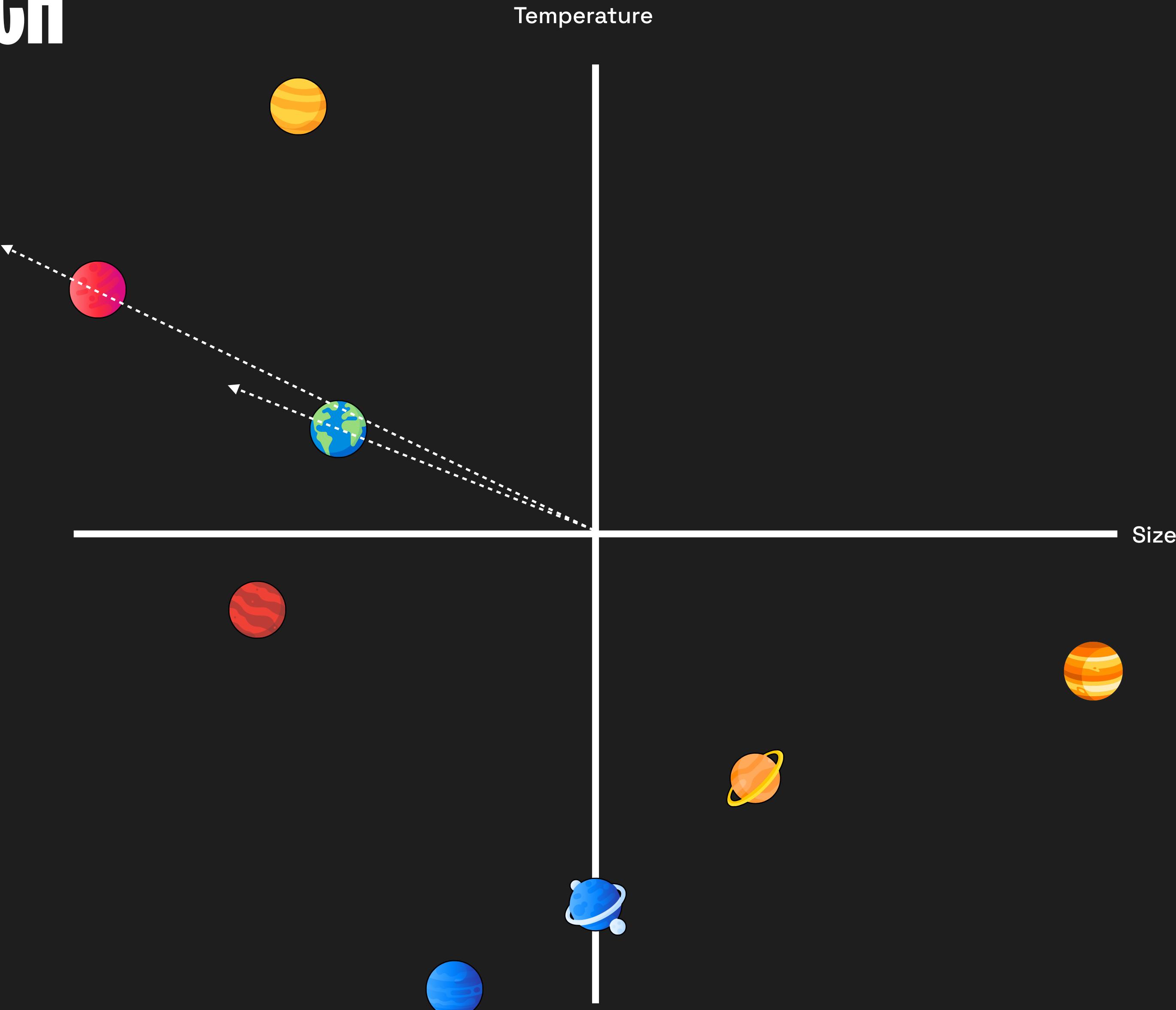
Planet	Size	Temp	Vector
Mercury	0,33	167	(-1, 0.5)
Venus	4,87	464	(-0.6, 1)
Earth	5,97	15	(-0.5, 0.2)
Mars	0,642	-65	(-0.7, -0.1)
Jupiter	1888	-110	(1.0, -0.3)
Saturn	568	-140	(0.3, -0.5)
Uranus	86.8	-195	(0, -0.9)
Neptune	102	-200	(-0.2, -1)



# VECTOR SIMILARITY SEARCH

How do we search?

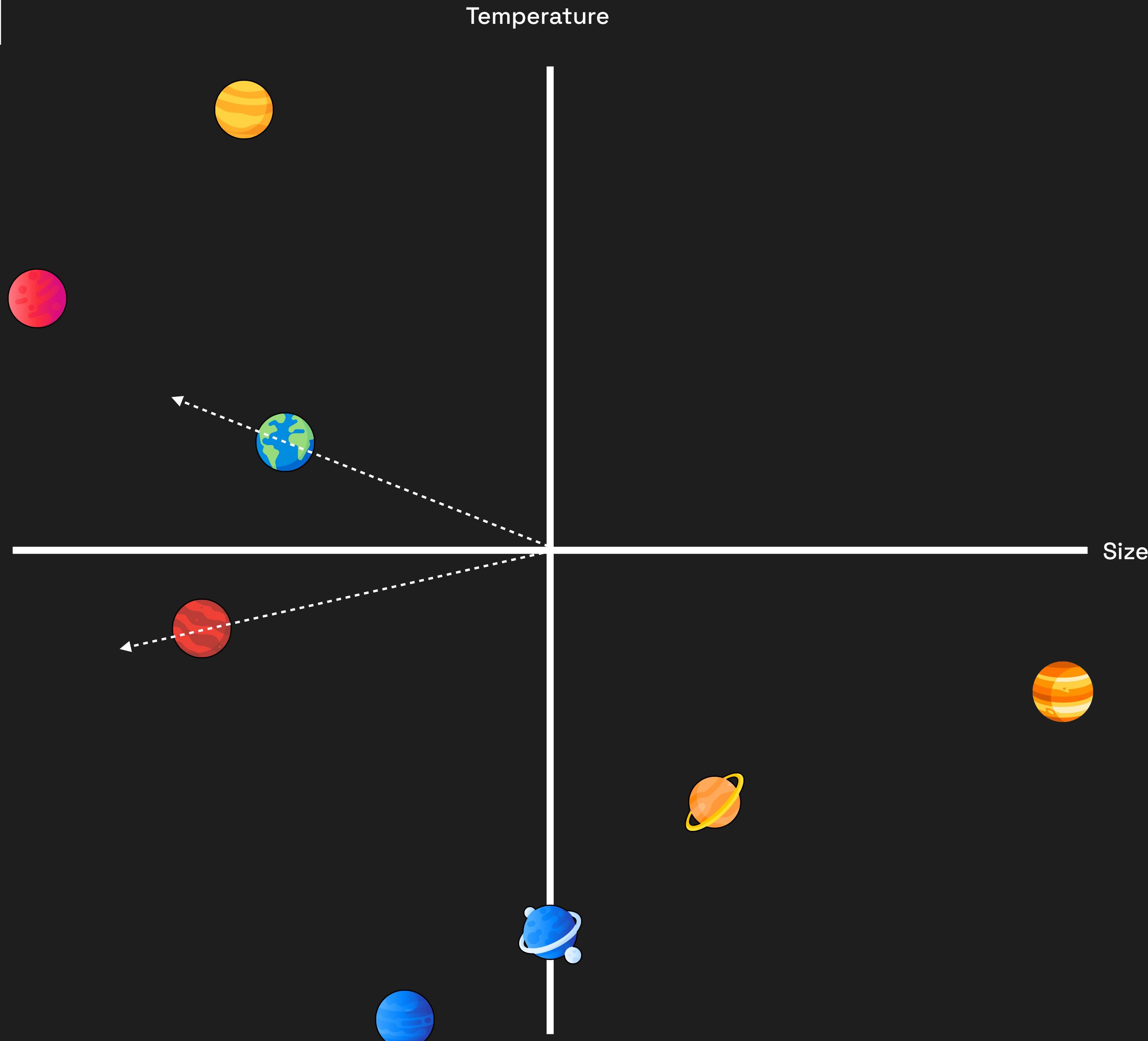
Cosine Similarity  
Angle Based



# VECTOR SIMILARITY SEARCH

How do we search?

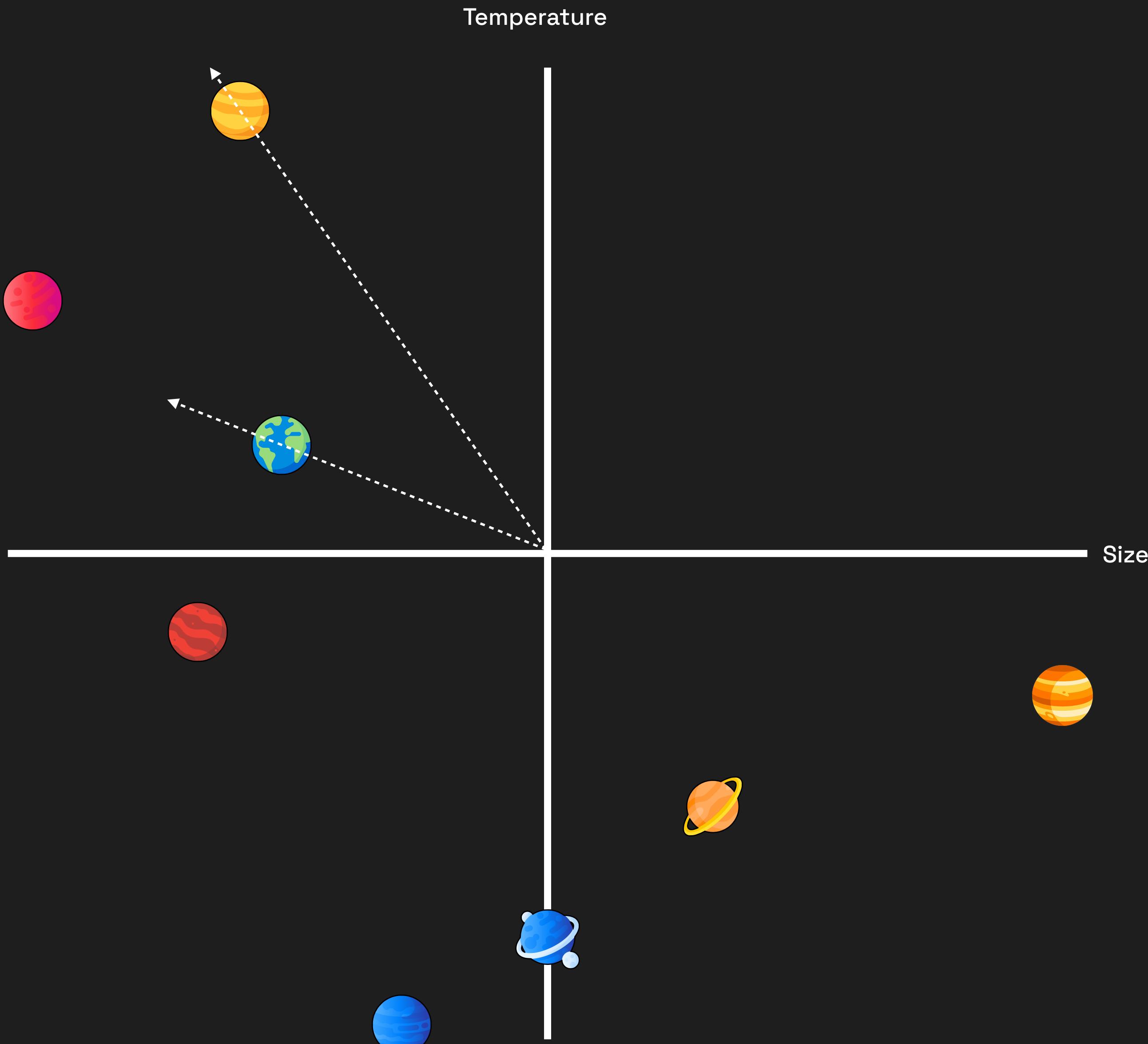
Cosine Similarity  
Angle Based



# VECTOR SIMILARITY SEARCH

How do we search?

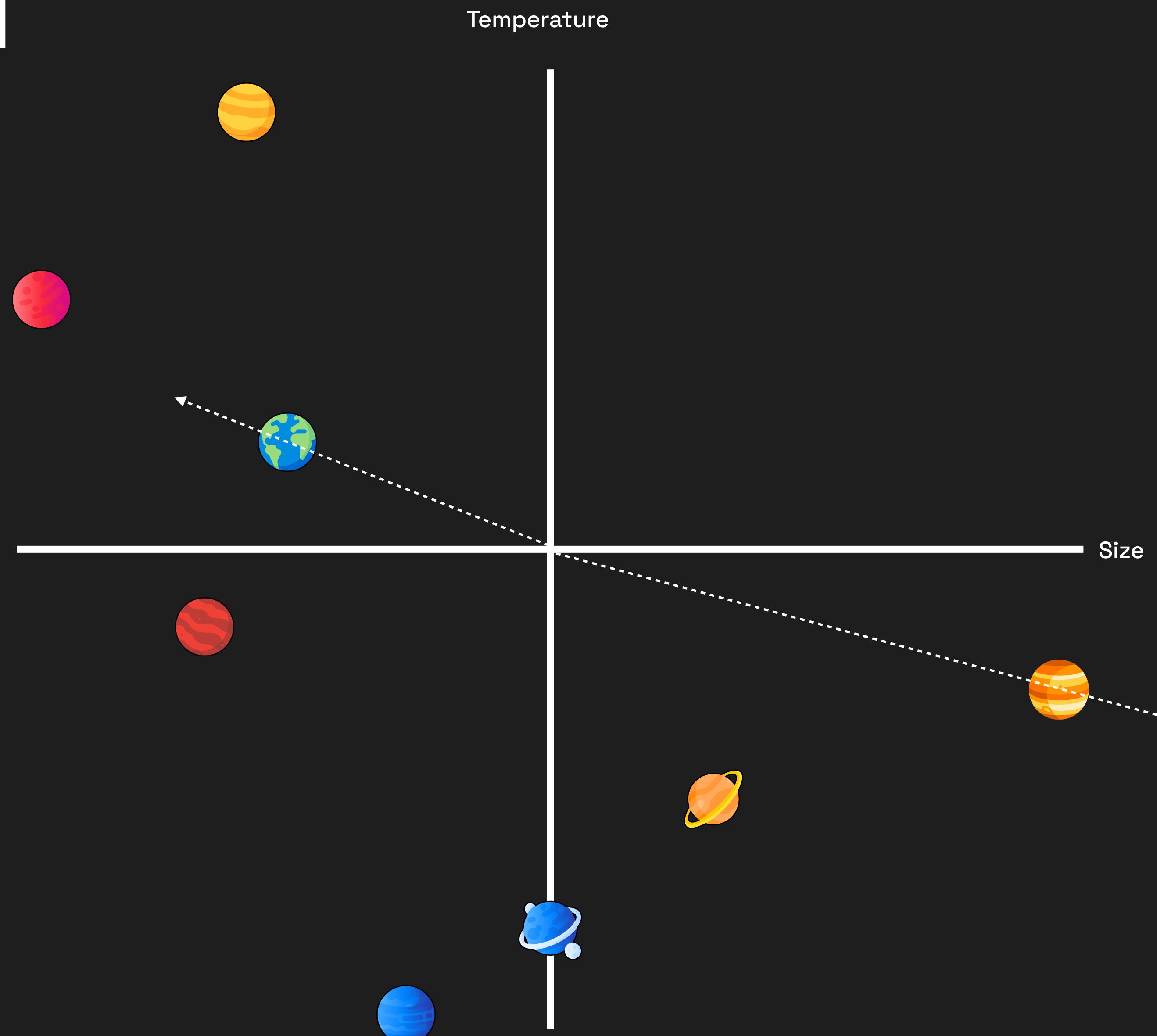
Cosine Similarity  
Angle Based



# VECTOR SIMILARITY SEARCH

How do we search?

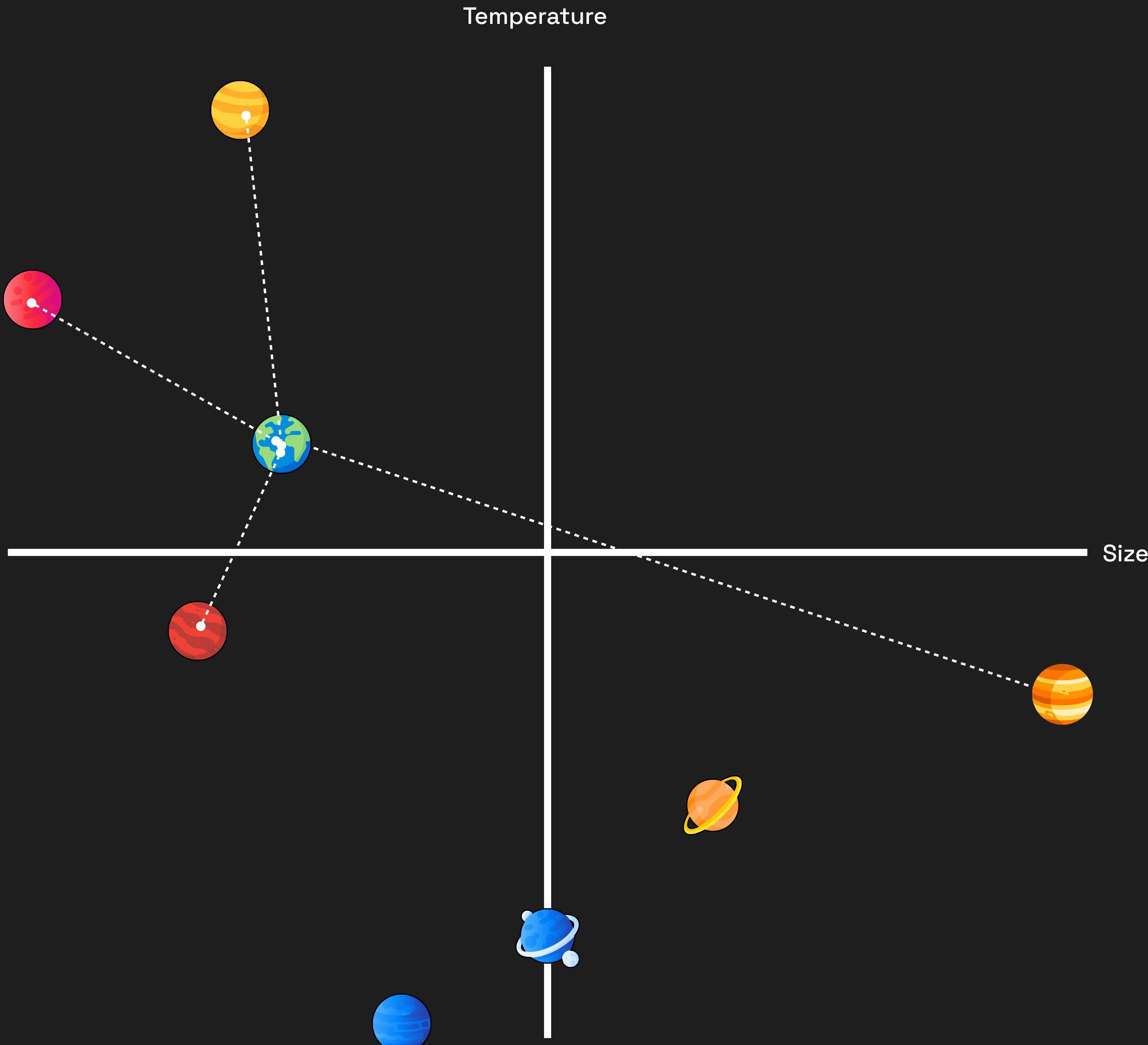
Cosine Similarity  
Angle Based



# VECTOR SIMILARITY SEARCH

How do we search?

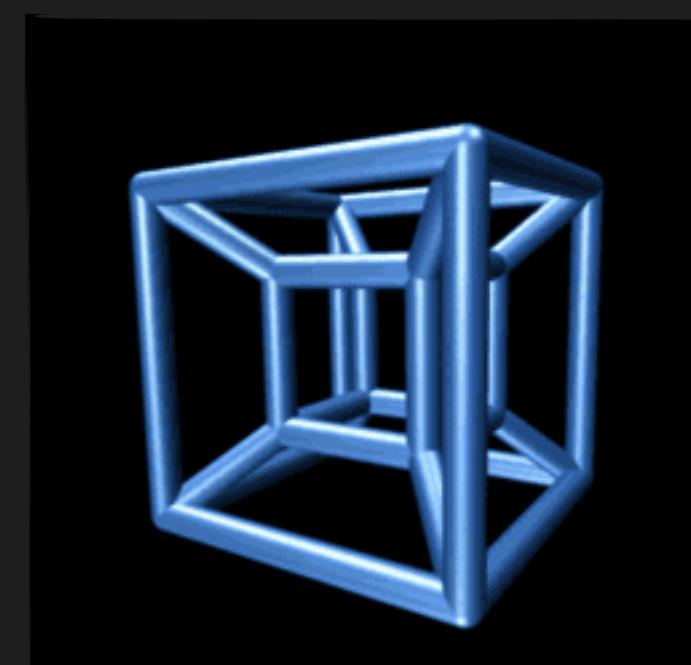
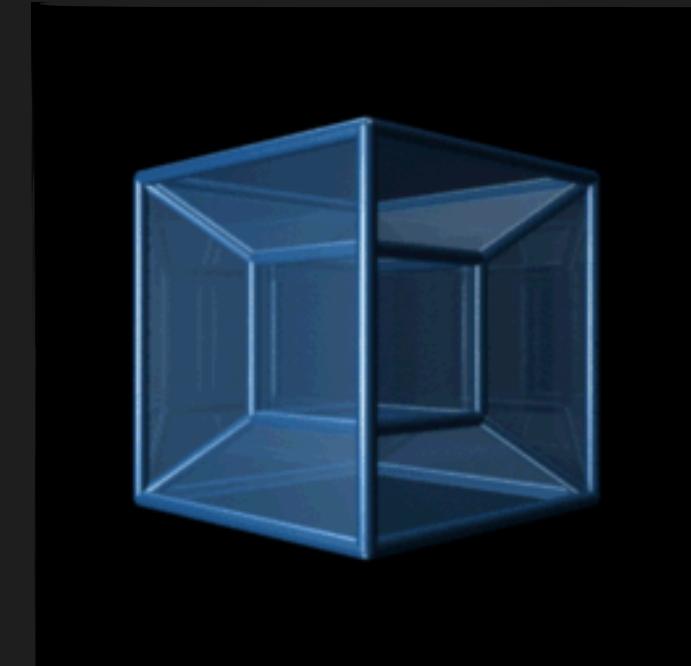
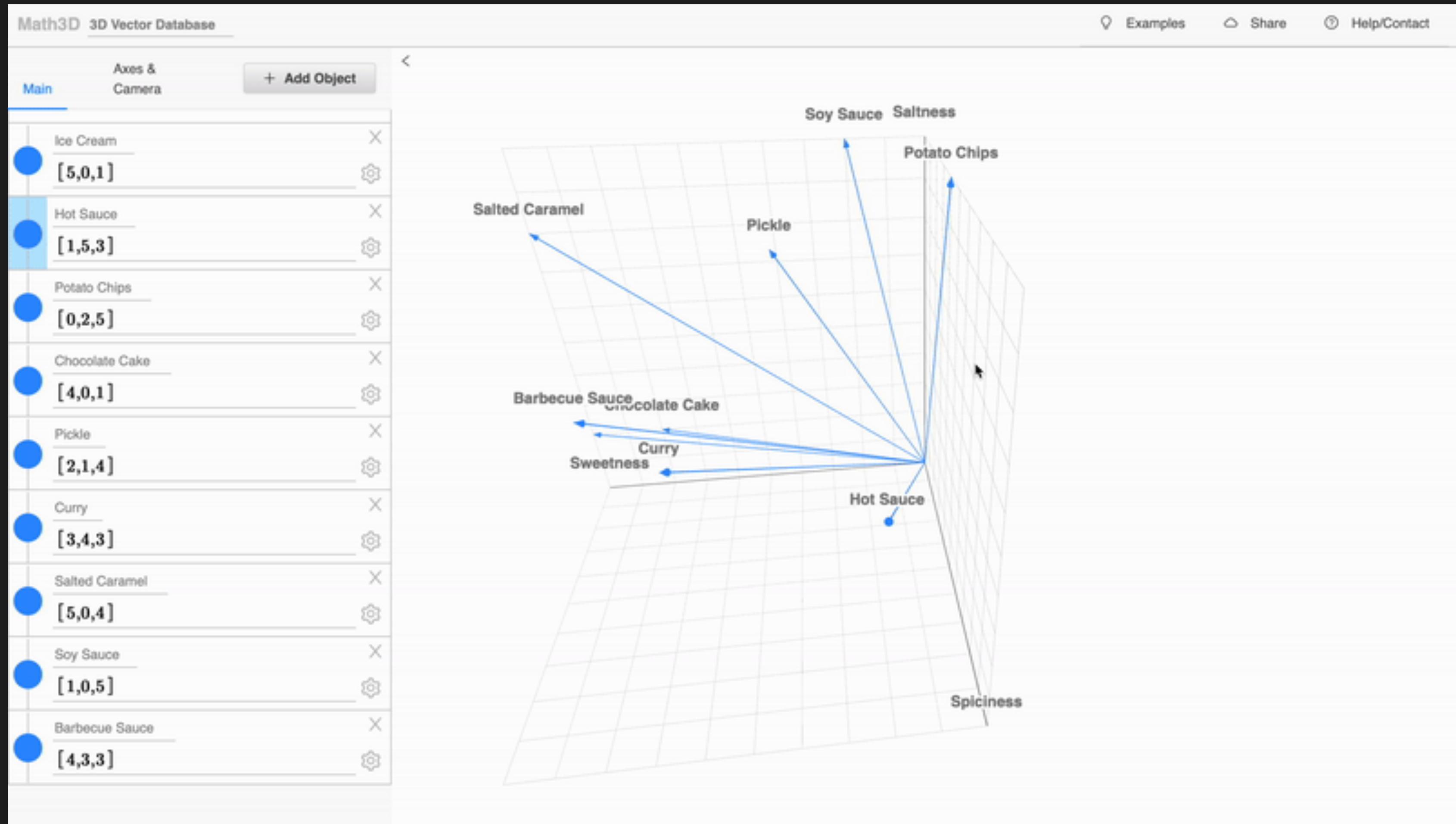
Euclidean Distance  
Magnitude Based



# VECTOR SIMILARITY SEARCH

## How many dimensions?

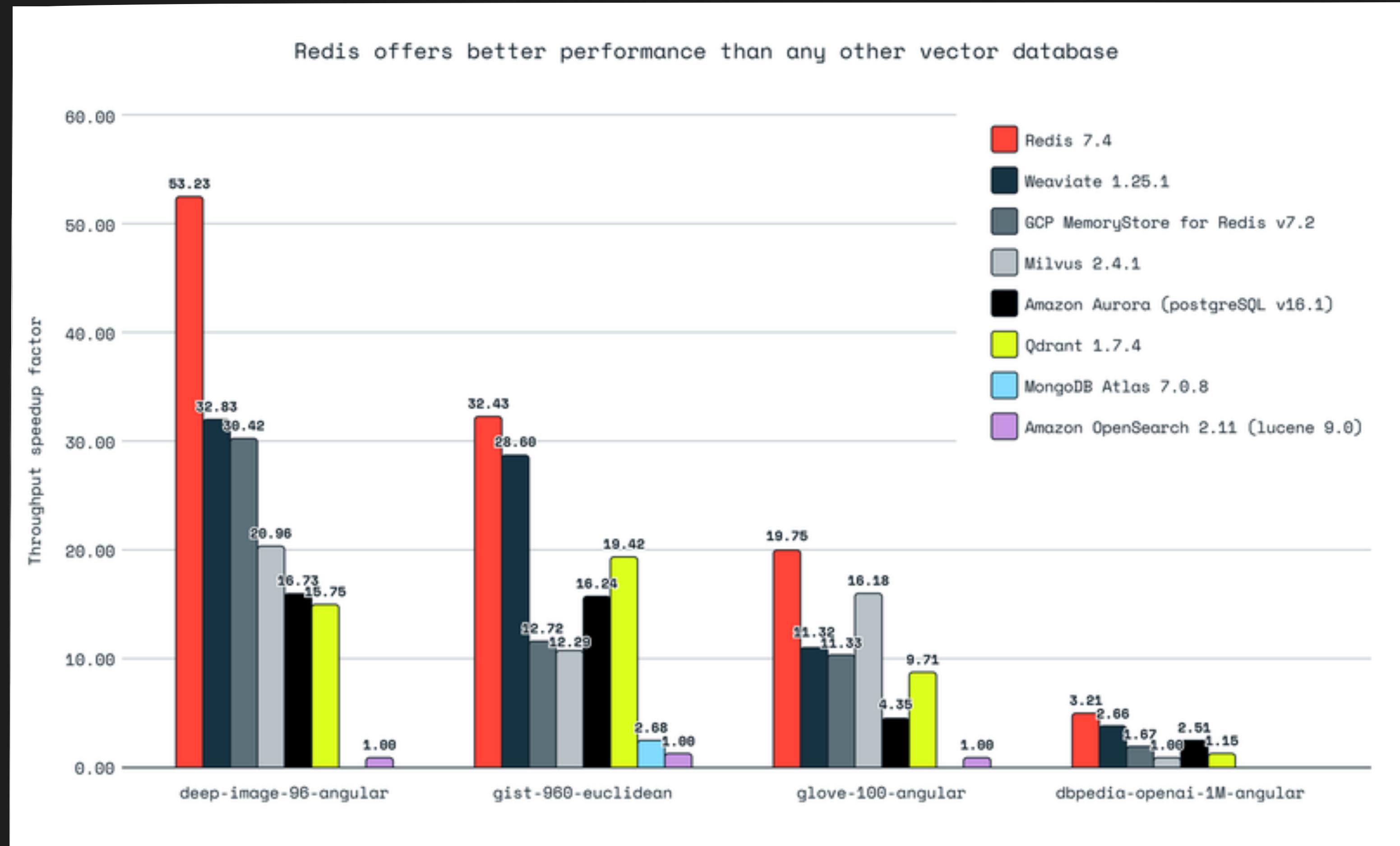
\*heads up de como escolher o embedding model



# VECTOR SIMILARITY SEARCH

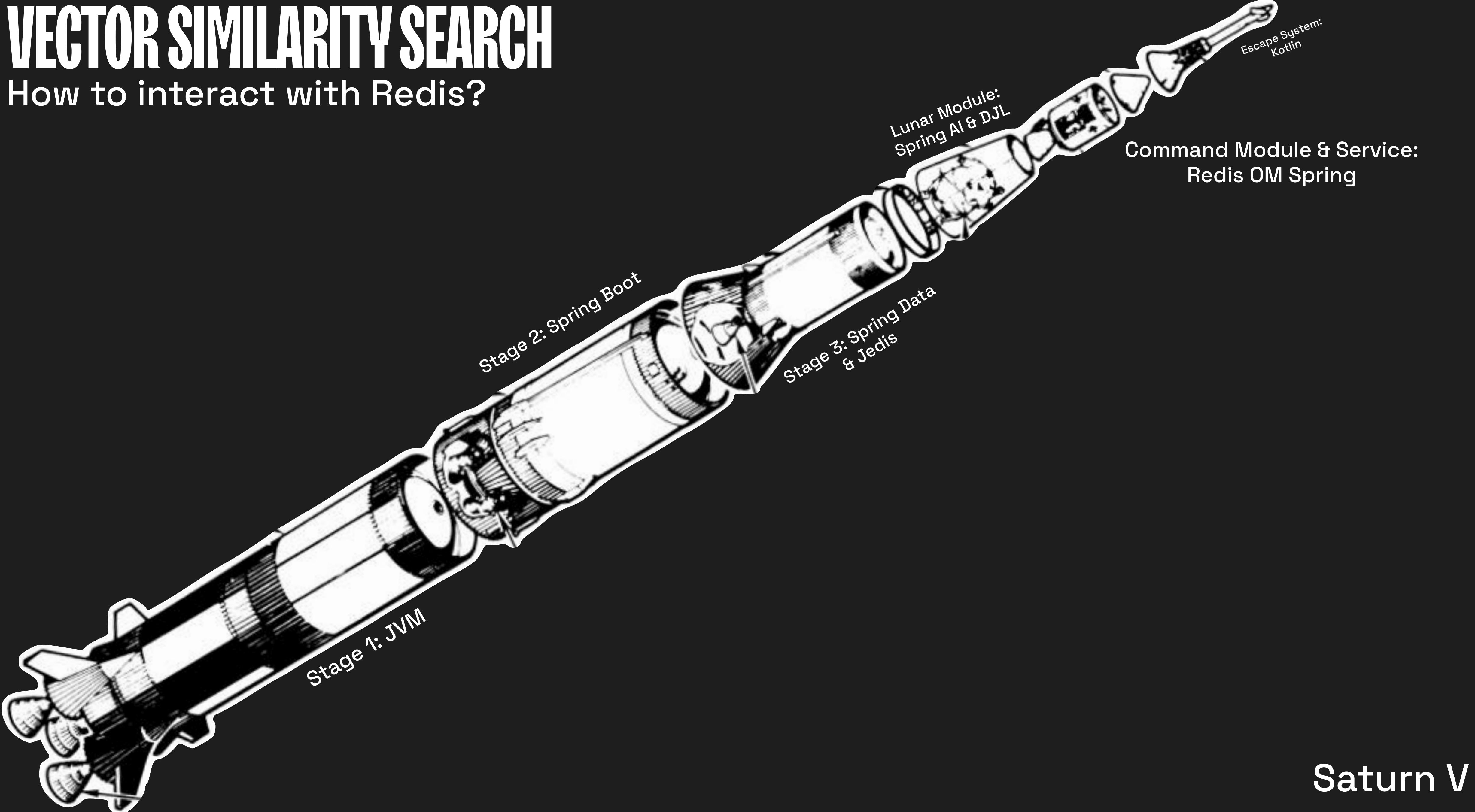
Where to store our vectors?

*Redis*



# VECTOR SIMILARITY SEARCH

How to interact with Redis?

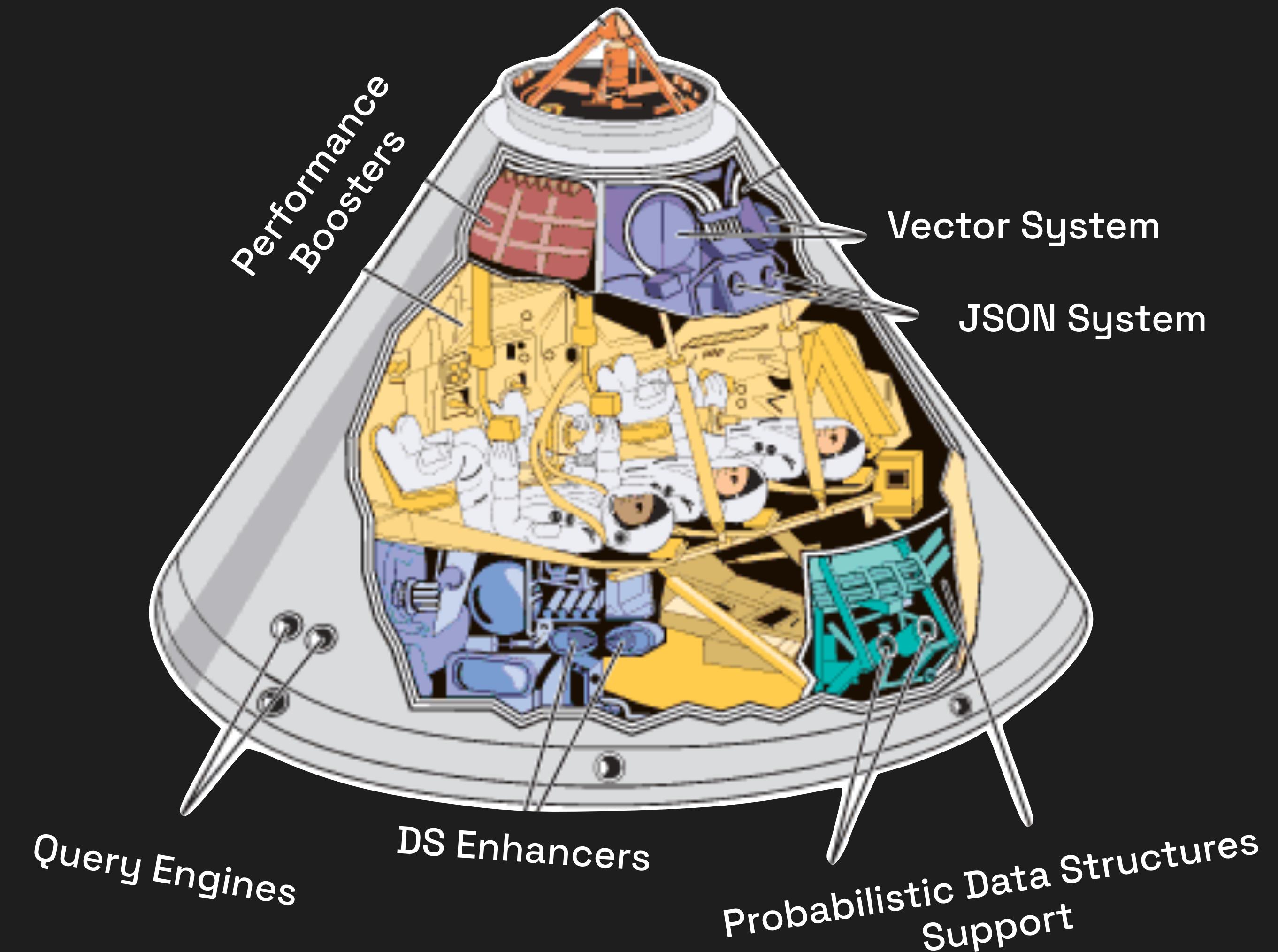


# VECTOR SIMILARITY SEARCH

How to interact with Redis?

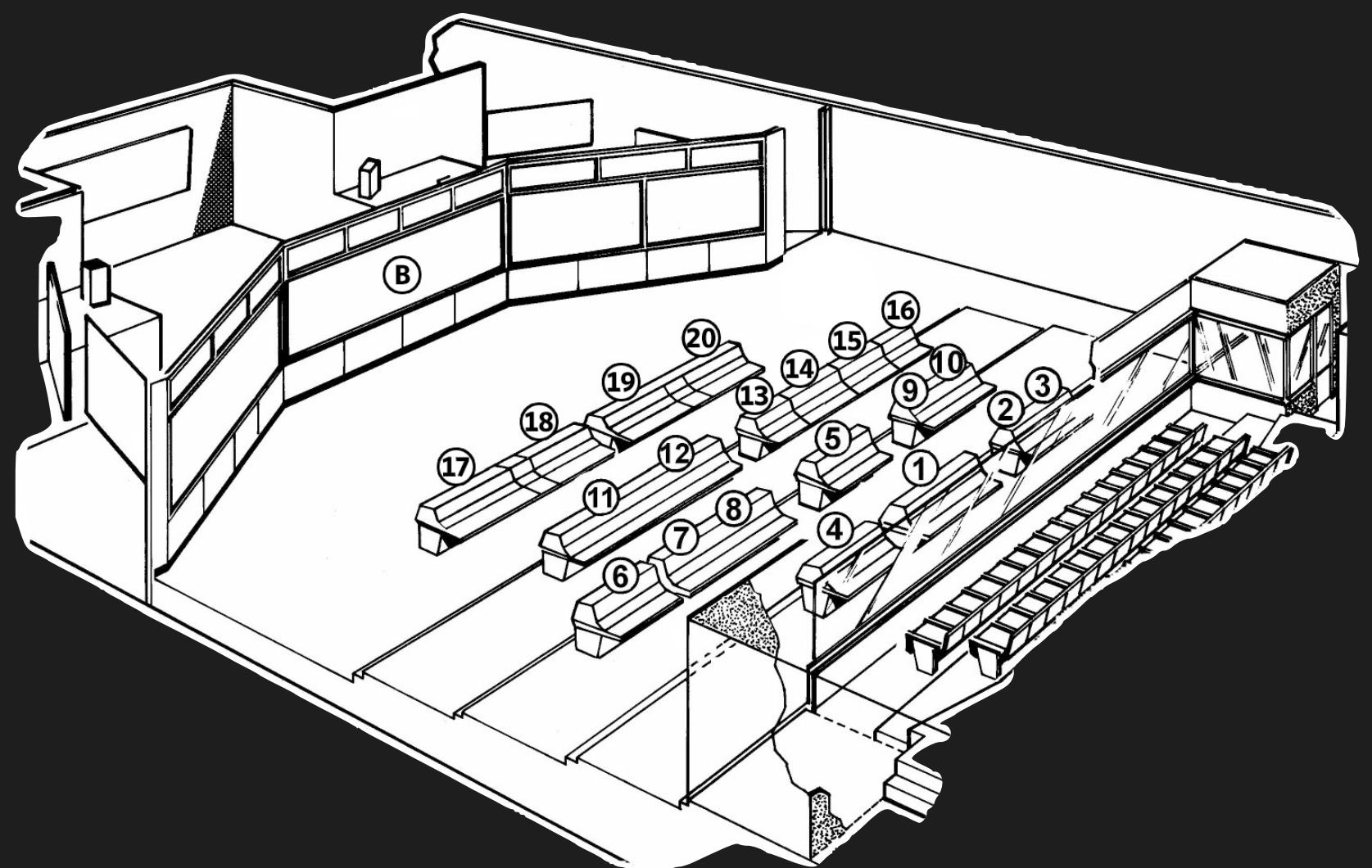
The Command Module

Redis OM Spring



# VECTOR SIMILARITY SEARCH

How to interact with Redis?



Mission Control

The Redis UI interface displays the following information:

**Database Scan Summary:** Results: 20 438. Scanned 20 438 / 20 438. Last refresh: 1 d.

**Key Types:** All Key Types

**Search Bar:** Filter by Key Name or Pattern

**Sorted Set:** utterances (No limit, 2 MB)

**CMSK-TYPE:** wordCount (No limit, 39 KB)

**Key Detail View:** HASH dev.raphaeldelio.redisapollo.hash.domain.Photograph:0565510

Field	Value	TTL
embeddedDescription	[ -0.03194248303771019, ... ]	No Limit
internalUrl	AS11-36-5395HR.jpg	No Limit
imagePath	classpath:static/images/...	No Limit
_class	dev.raphaeldelio.redisap...	No Limit
name	AS11-36-5395	No Limit
timestamp	0565510	No Limit
embeddedImage	[ 2.6660306453704834, 0... ]	No Limit
description	Another view through th...	No Limit
externalUrl		No Limit

**Footer:** CLI Command Helper Profiler Let us know what you think

**TAKING OFF**

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

# COLLECTING THE DATA

```
1 {
2   [
3     "1092339",
4     "Armstrong",
5     "I'm at the foot of the ladder. The LM footpads are only depressed in the surface about 1 or 2 inches,
6     although the surface appears to be very, very fine grained, as you get close to it. It's almost like a powder.
7     - ground mass is very fine",
8     "T"
9   ],
10  [
11    "1092414",
12    "Armstrong",
13    "Okay. I'm going to step off the LM now.",
14    "T"
15  ],
16  [
17    "1092426",
18    "Armstrong",
19    "That's one small step for man, one giant leap for Mankind.",
20    "T"
21  ]
22 }
```

# LOADING THE DATA

```
1 @Data  
2 @RedisHash ← Annotation for defining a HASH object  
3 public class Utterance {  
4  
5     @Id  
6     private String timestamp;  
7  
8     @Indexed  
9     @NonNull  
10    private String speaker;  
11  
12    @Indexed ← Annotation for allowing efficient querying  
13    @NonNull  
14    private String text;  
15  
16    @Indexed  
17    @NonNull  
18    private String speakerId;  
19 }
```

Creating the repository

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

# VECTORIZING THE DATA

```
1  @NonNull  
2  @Vectorize( ← Data to be vectorized  
3      destination = "embeddedText",  
4      embeddingType = EmbeddingType.SENTENCE  
5  )  
6  private String text;  
7  
8  @Indexed( ← Field where the vectorize  
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;
```

# 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)) ← Searching for the 3 nearest neighbors  
6     .sorted(Utterance$.EMBEDDED_TEXT_SCORE)  
7     .map(Fields.of(Utterance$.THIS, Utterance$.EMBEDDED_TEXT_SCORE))  
8     .collect(Collectors.toList());
```

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

```
1 {
2   [
3     "109:19:18",
4     "1",
5     "On the porch"
6   ],
7   [
8     "109:22:50",
9     "2",
10    "Coming down the ladder"
11  ],
12  [
13    "109:24:26",
14    "2",
15    "First step"
16  ],
17  [
18    "109:30:54",
19    "2",
20    "Armstrong reminded to get a contingency sample"
21  ]
22 }
```

# SUMMARIZATION

```
1 public void summarize(boolean overwrite) {
2     logger.info("Summarizing TOC entries");
3     List<TOCData> tocDataList = tocDataRepository.findAll();
4
5     for (TOCData tocData : tocDataList) {
6         ChatResponse response = chatModel.call(
7             new Prompt(List.of(
8                 new SystemMessage("""
9                     You are a helpful assistant who summarizes utterances of the Apollo 11 mission.
10                    Make these summaries very dense with all curiosities included.
11                    Limit the summary to 512 words.
12                    """),
13                 new UserMessage(tocData.getConcatenatedUtterances())
14             )));
15     };
16
17     tocData.setSummary(response.getResult().getOutput().getContent());
18     tocDataRepository.save(tocData);
19     logger.info("Summarized TOC entry: {}", tocData.getStartDate());
20 }
21
22 logger.info("Summarized TOC entries");
23 }
24 }
```

# EXTRACTION

```
1 logger.info("Generating questions for TOC entries");
2 List<TOCData> tocDataList = tocDataRepository.findAll();
3 for (TOCData tocData : tocDataList) {
4     ChatResponse response = chatModel.call(
5         new Prompt(List.of(
6             new SystemMessage("""
7                 You are a helpful assistant that is helping me predict which questions can be asked by
8                 people who are trying to rediscover the Apollo 11 mission data. You will be given a number of
9                 utterances and you will predict the questions that can be asked by people who are trying to
10                rediscover the Apollo 11 mission data. You will ONLY return the questions separate by breaklines,
11                and nothing more. You will NEVER return more than 512 words.
12            """),
13            new UserMessage(tocData.getConcatenatedUtterances())
14        )));
15    );
16
17    var questions = Arrays.stream(response.getResult().getOutput().getContent().split("\n"))
18        .filter(q →!q.isBlank()).toList();
19    tocData.setQuestions(questions);
20    tocDataRepository.save(tocData);
21    logger.info("Generated questions for TOC entry: {}", tocData.getStartDate());
22 }
23 logger.info("Generated questions for TOC entries");
```

# QUERVING

Searching on the questions instead of texts

```
1 SearchStream<UtteranceQuestions> stream = entityStream.of(UtteranceQuestions.class);
2 List<Pair<UtteranceQuestions, Double>> questionsAndScores = stream
3     .filter(UtteranceQuestions$.EMBEDDED_QUESTION.knn(3, embeddingToBeQueried))
4     .sorted(UtteranceQuestions$.EMBEDDED_QUESTION_SCORE)
5     .map(Fields.of(UtteranceQuestions$.THIS, UtteranceQuestions$.EMBEDDED_QUESTION_SCORE))
6     .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

```
1 @PostMapping("/search-by-question")
2 public Map<String, Object> searchByQuestion(@RequestBody Map<String, String> requestBody) {
3     String queryText = requestBody.get("query");
4     logger.info("Received question: {}", queryText);
5
6     byte[] embedding = embedder.getTextEmbeddingsAsBytes(List.of(queryText), UtteranceQuestions$.QUESTION).get(0);
7     SearchStream<UtteranceQuestions> stream = entityStream.of(UtteranceQuestions.class);
8     List<Pair<UtteranceQuestions, Double>> questionsAndScores = stream
9         .filter(UtteranceQuestions$.EMBEDDED_QUESTION.knn(3, embedding))
10        .sorted(UtteranceQuestions$.EMBEDDED_QUESTION_SCORE)
11        .map(Fields.of(UtteranceQuestions$.THIS, UtteranceQuestions$.EMBEDDED_QUESTION_SCORE))
12        .collect(Collectors.toList());
13
14     List<Map<String, String>> matchedQuestions = questionsAndScores.stream()
15        .map(pair → {
16            UtteranceQuestions question = pair.getFirst();
17            return Map.of(
18                "question", question.getQuestion(),
19                "utterances", question.getUtterancesConcatenated(),
20                "score", pair.getSecond().toString()
21            );
22        })
23        .collect(Collectors.toList());
24
```

```
25 ChatResponse response = chatModel.call(
26     new Prompt(List.of(
27         new SystemMessage("""
28             You are an expert assistant specializing in the Apollo missions. Your goal is to provide accurate,
29             detailed, and concise answers to user inquiries by utilizing the provided Apollo mission data.
30             Rely solely on the information given below and avoid introducing external information.
31             """),
32         new SystemMessage("Apollo mission data: " + questionsAndScores.stream()
33             .map(it → it.getFirst().getUtterancesConcatenated())
34             .collect(Collectors.joining("\n"))),
35         new UserMessage("User question: " + queryText)
36     ))
37 );
38
39 logger.info("AI response: {}", response.getResult().getOutput().getContent());
40 return Map.of(
41     "query", queryText,
42     "answer", response.getResult().getOutput().getContent(),
43     "matchedQuestions", matchedQuestions
44 );
45 }
```

# DEMO

# TAKEAWAYS

- RUNNING REDIS & REDIS INSIGHT ON DOCKER**  16:50 How to start Redis locally with Docker & Get started with... Coding with Raphael De Lio
- UNDERSTANDING PERSISTENCE IN REDIS**  19:42 Understanding Redis Persistence - AOF & RDB +... Coding with Raphael De Lio
- UNDERSTANDING TRANSACTIONS IN REDIS**  10:53 Understanding Transactions In Redis (Getting Started) Coding with Raphael De Lio
- USING CHAT GPT TO BUILD A KOTLIN + REDIS APPLICATION FROM SCRATCH**  26:38 Using ChatGPT to build me a Kotlin + Redis Application fro... Coding with Raphael De Lio
- UNDERSTANDING REDIS PUB/SUB**  9:33 Understanding Redis Pub/Sub (Getting Started) Coding with Raphael De Lio
- UNDERSTANDING REDIS STREAMS**  15:59 Understanding Redis Streams Coding with Raphael De Lio
- RATE LIMITERS THE RIGHT WAY**  5:42 Rate limiting with Redis: An essential guide Redis
- THE SIMPLEST RATE LIMITER**  6:39 Fixed Window Counter Rate Limiter (Redis & Java) Redis
- MAXIMUM BURST FLEXIBILITY**  7:39 Token Bucket Rate Limiter (Redis & Java) Redis

# RAPHAEL DE LIO

## DEVELOPER ADVOCATE



Redis

