Benchmarking Vector Databases on Code Embeddings

CS 854 – Performance Engineering

Fall 2023, Ali Mashtizadeh

Vikram **Subramanian**, Raymond **Chang**, Yahya **Jabary**

N-dimensional, float

Benchmarking Vector Databases on Code Embeddings

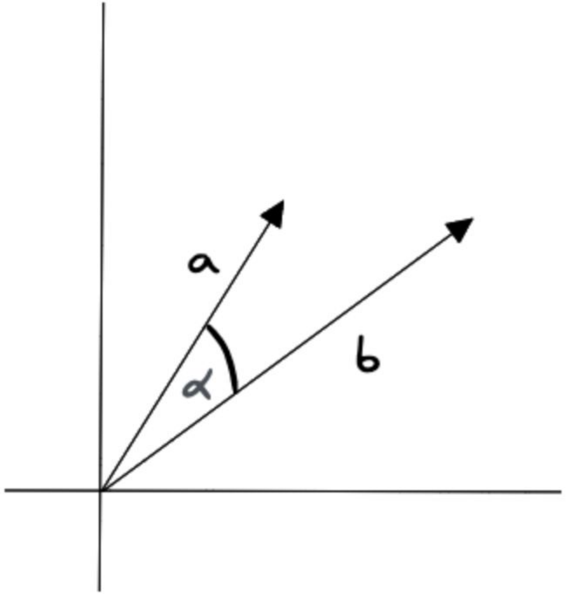
Vector representations of documents

What are vector databases?

- Specifically designed to store and retrieve vectors
- Given query vector ***q [0, 0.3, 0.1]***
  - Retrieves **approximately** similar vectors to ***q***
  - Similarity depends on DB **load**
- Ex: MilvusDB, ChromaDB

How do they work? - Similarity

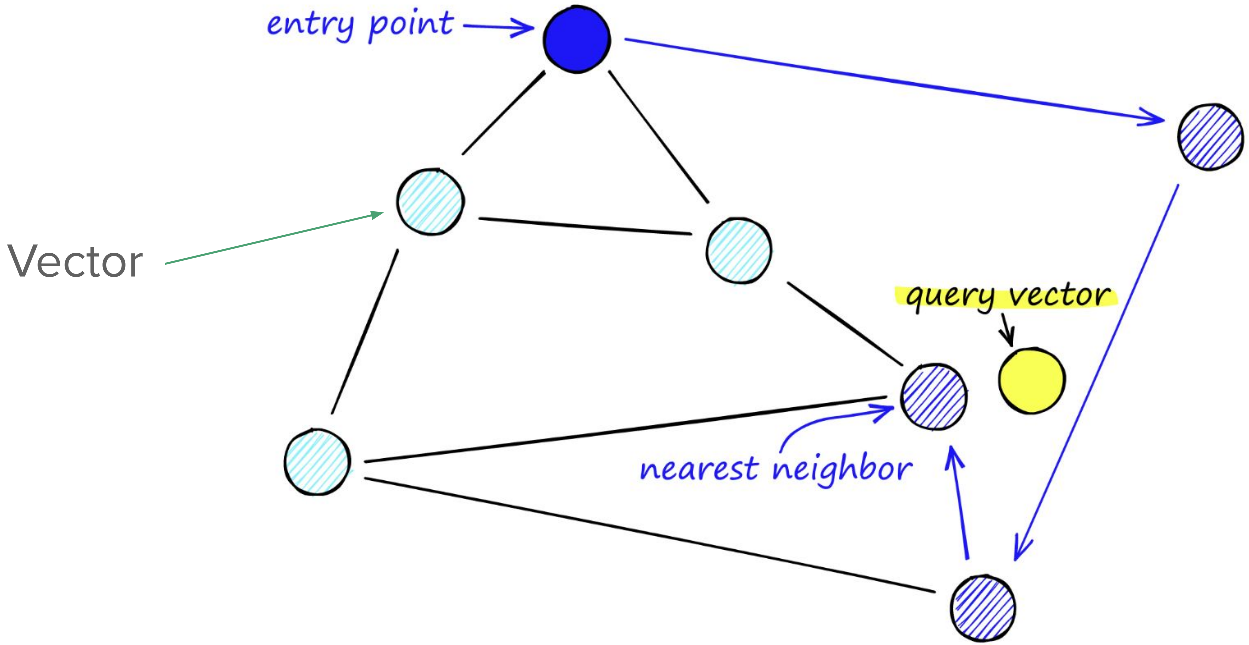
Cosine Similarity

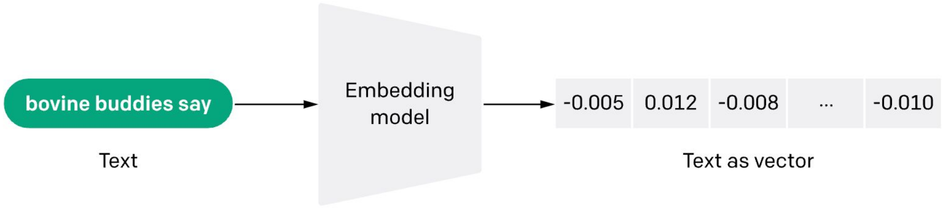


$$sim(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}|| \cdot ||\mathbf{b}||}$$

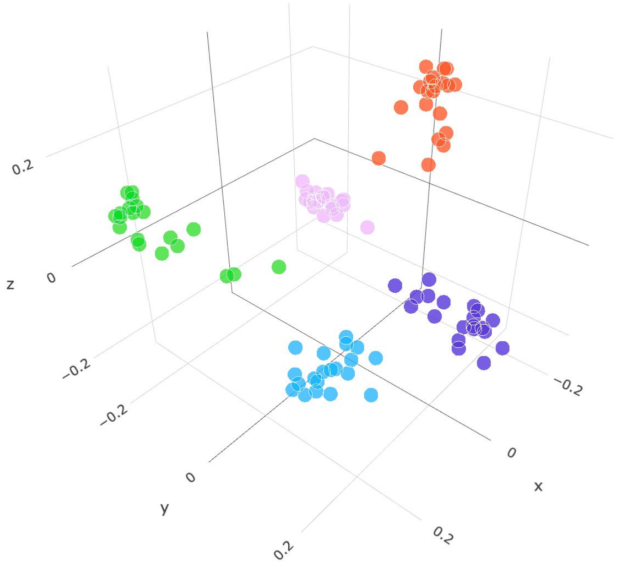
How do they work? - Indexing

Navigable Small Worlds





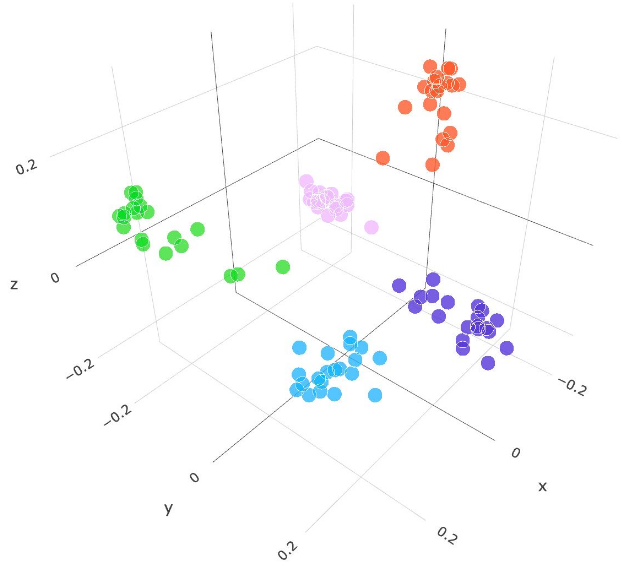
bovine buddies say
Text
Embedding model
-0.005
0.012
-0.008
...
-0.010
Text as vector



z
0.2
0
−0.2
y
0
0.2
−0.2
x
0
0.2
−0.2

Why should I care?

- **Novelty**:
    - No standard test dataset for code
    - New Vector DBs every month
- **Practicality**: used in many AI/ML apps
    - Searching and clustering files of code by semantics



What did we do?

Contributions:

- A very large high-dimension (1024 dimension) dataset
- First code benchmarking dataset
- Some basic benchmarking

Why aren't there are many large high-dimensional datasets?

- Computationally expensive
  - 5*10^5 functions in our dataset and we use an embedding of 1024
  - We have 3*10^9 floats- We need to do 3*10^18 operations
  - Will take 1000 H100s, 3 days
- Solution: Precompute nearest neighbors by generating mutations of code blocks

Step 1)
Getting initial dataset

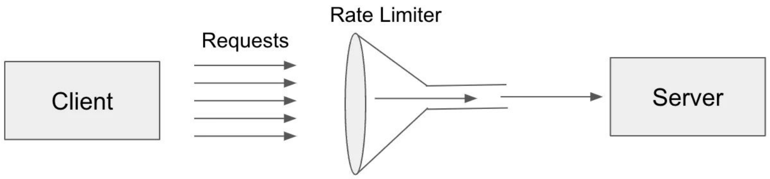
Github API

Goal: 500.000+ files of code

Github API rate limits:

a. anonymous: 60 requests/h ⟶ +8,333h
b. authenticated: 5000 requests/h ⟶ +100h
c. enterprise: 15000 requests/h ⟶ +33h (and lots of money)

How does rate limiting work?



By identifying you through your:

- IP address, geolocation (default)
- API auth key
- User session cookies (mostly Akamai)
- HTTP requests (mostly Cloudflare)
- TLS fingerprints (very rare)

Rotating IPs via XFF vulnerability

Rotating proxies for each request through:

a. AWS API Gateway's large IP pool
b. BurpSuite PortSwigger IProtate

HTTP Request:

```
GET /api/v1/otp/check HTTP/1.1
Host: vulnerable-website.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 9
X-Forwarded-For: <client>, <proxy1>, <proxy2>
```

Result: **Unsuccessful**

- Github detects popular proxy pool headers like "`X-Amzn-Trace-Id`"
- We don't want to pay for custom IP pools

Fake resource paths

```
No semantics: %00, %09, %0a, %0c, %20, …
Bare endpoint: /api/v4/endpoint

/api/v4/endpoint
/api/v4/Endpoint
/api/v4/EndPoint
/api/v4/endpoint%00
/api/v4/%0aendpoint
/api/v4/endpoint%09
/api/v4/%20endpoint

Fake query params:
/api/v4/%20endpoint?phone=+17342239011&code[]=123456&code[]=654321&...&code[]=331337
```

Result: Unsuccessful

Token rotation / Overwriting ETags

```
$ curl -i https://api.github.com/users/defunkt
> HTTP/2 200
> ...
> etag: W/"61e9...f26f0"


$ curl -i -H 'If-None-Match: "61e9...f26f0"' \
                              https://api.github.com/users/sueszli
> HTTP/2 304
```

Result: Unsuccessful

Bandwidth throttling

```
from ratelimit import limits, RateLimitException
from backoff import on_exception, expo

import requests


@on_exception(expo, RateLimitException, max_tries=8)
@limits(calls=15, period=900)                                  # <-- limit requests
def call_api(url):
    response = requests.get(url)

    if response.status_code != 200:
        raise Exception('API response: {}'.format(response.status_code))
    return response
```

Result: **Unsuccessful** – now we're **even slower**

Scraping the Github “search” page

- Lets you query by language **and repository size**
- Only displays 5 pages when you get rate limited
- Scraped around 500 entries manually

Result: Unsuccessful

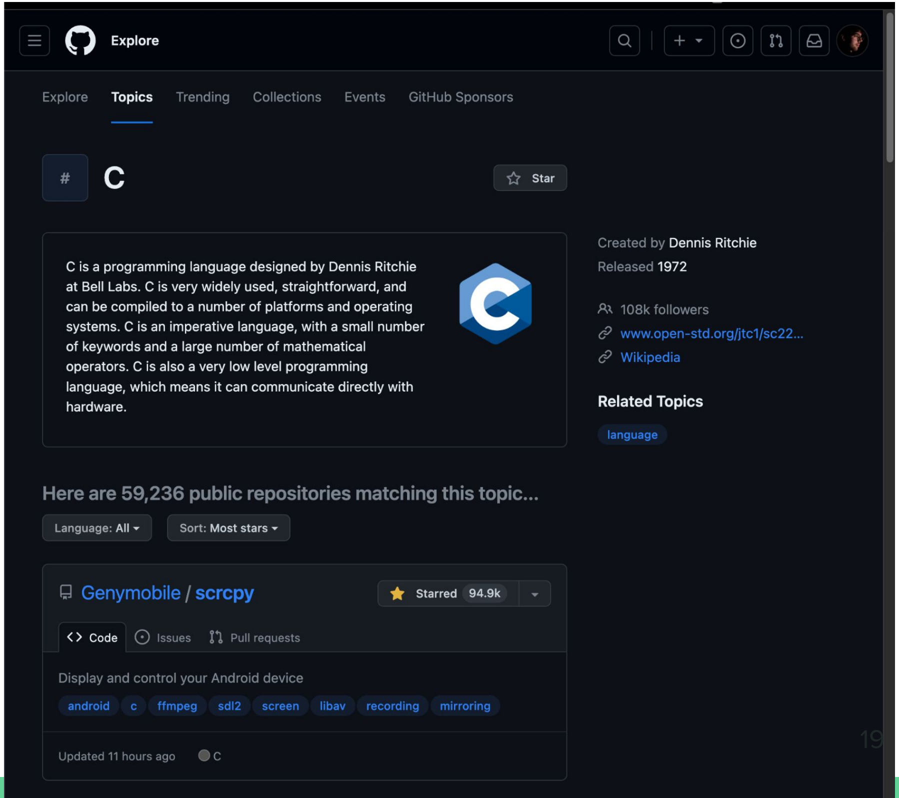
Github, why?

Scraping the Github “topics” page

- Novel discovery:
  Single page scraping

- 2000 repositories per page
  or ~15 repos per second

Result: Very Successful



Final steps

1. Scrape links to repositories through “Github topics”
   Total repositories: 3,494
2. Clone
3. Filter out non-code
   Total files: 649,257
4. Chunk into 1,000 file commits (>600 commits in ~2 days)
   Otherwise you have to pay for Github LFS

Step 2)
Mutating data to generate clusters

Mutation

- Must generate semantically similar code blocks
- Mutation:
  - Insert block of dead code
- For each function generate five mutated versions
  - Mutations increase in size

Mutation Example

Original

```
def foo():
    ...
```

Mutations

1. def foo():
```
        if False:
            i = 0
        ...
```
2. def foo():
```
        if False:
            i = 0
            while(false):
                ...
        ...
```

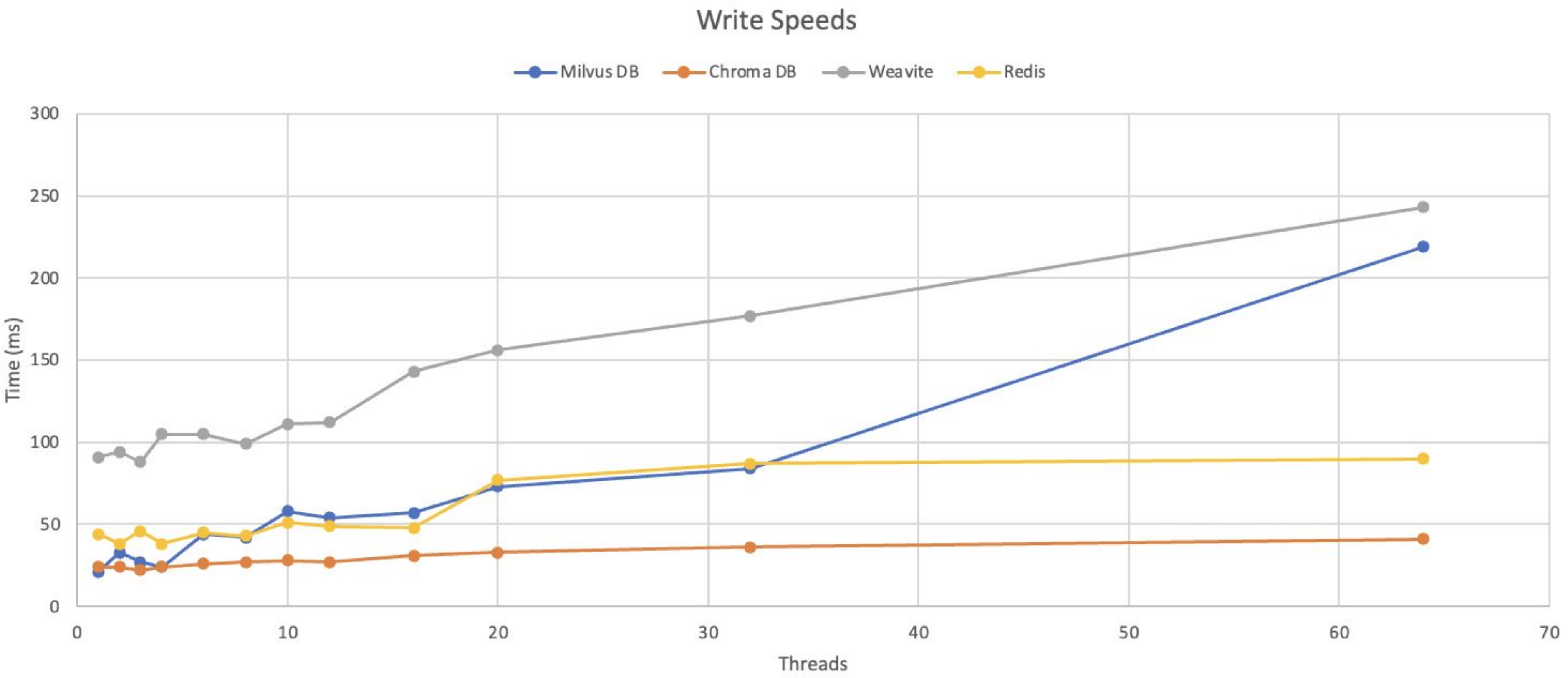
...

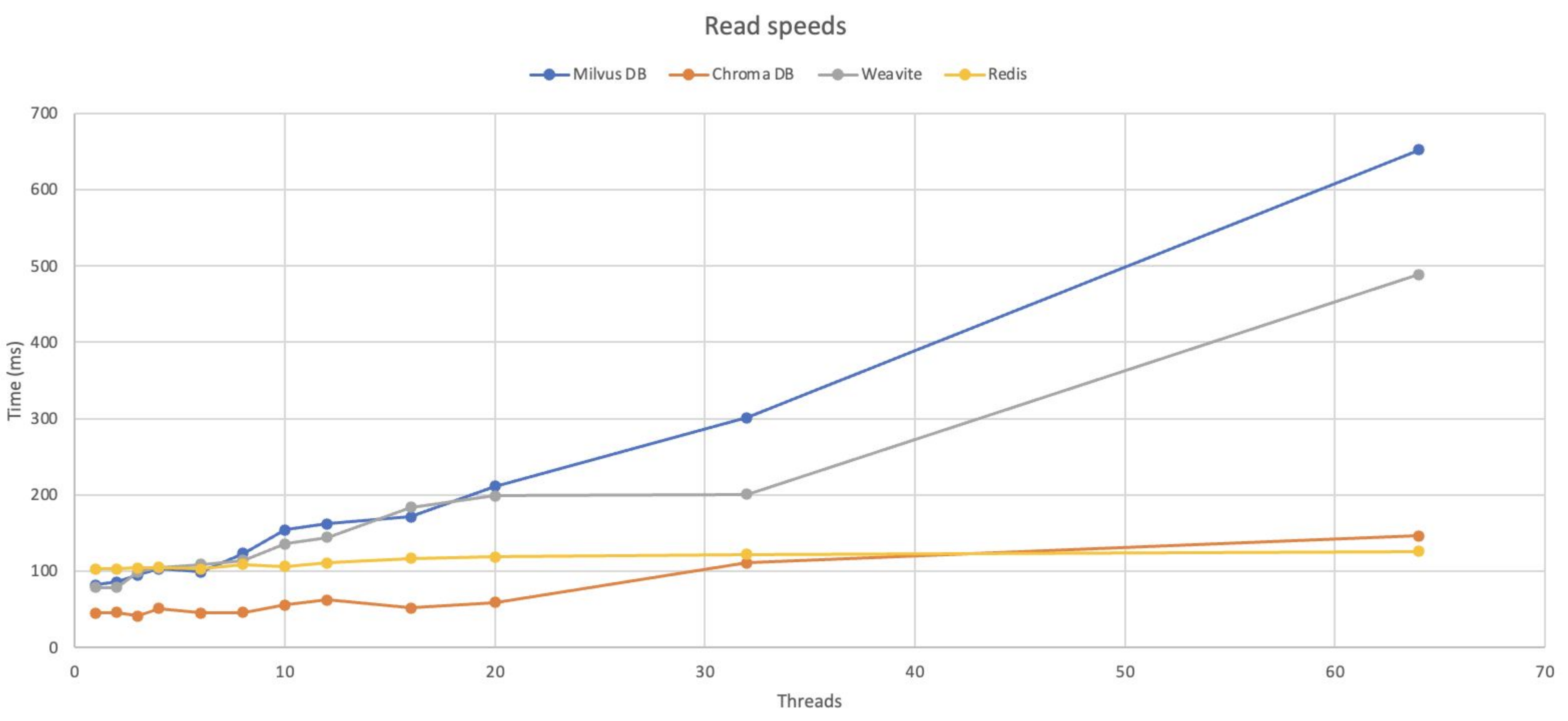
JSON

Mutation analysis

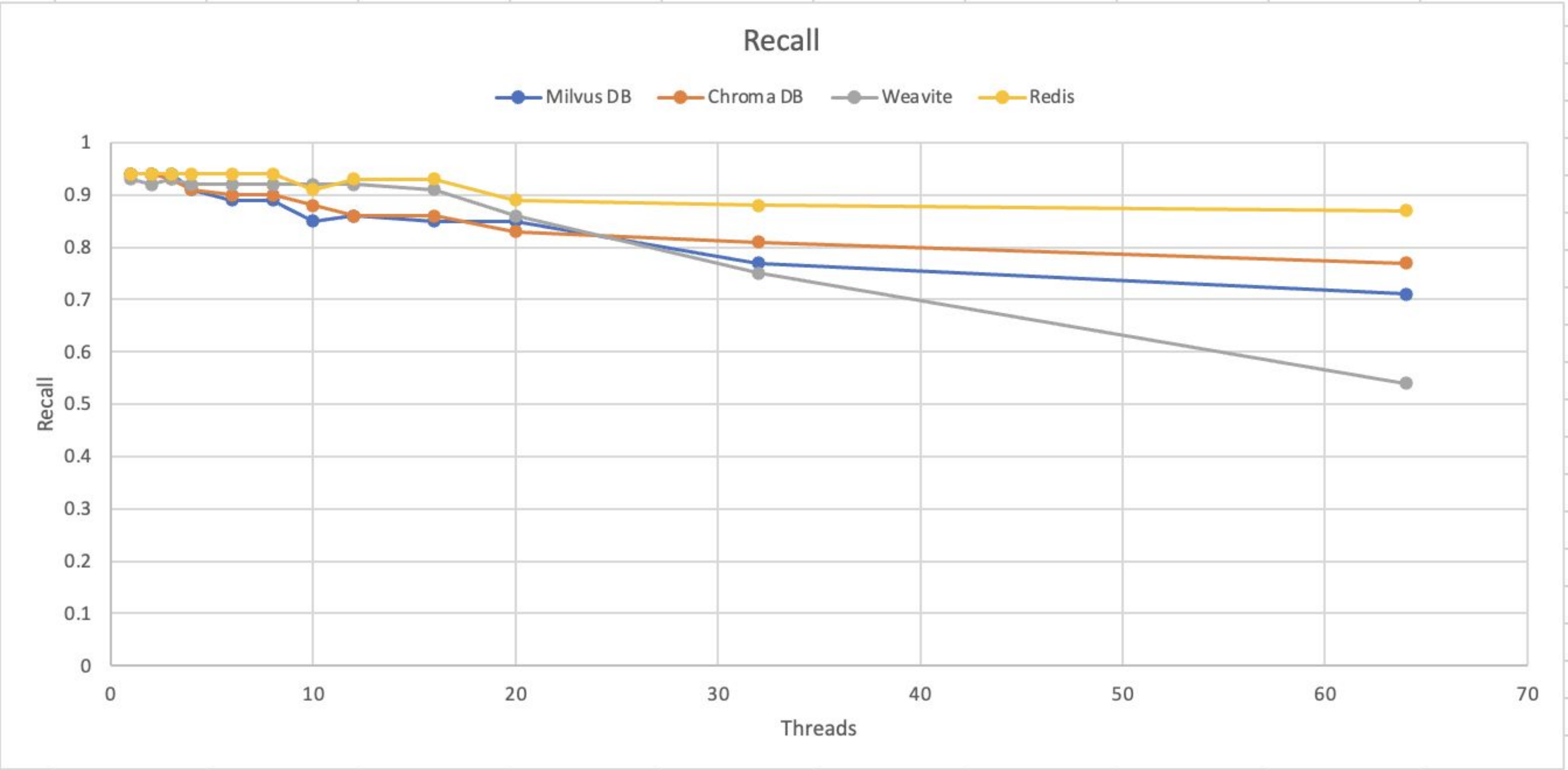
- Want Recall vs db load
- Must ensure mutations actually decrease similarity
- Compare
  - Similarity of the two largest
    - 0.68
  - Smallest function: Similarity with mutations
    - 0.82 to 0.92
  - Largest function: Similarity with mutations
    - 0.85 to 0.91

Step 3) Benchmarks


Write Speeds
Milvus DB
Chroma DB
Weavite
Redis
Time (ms)
300
250
200
150
100
50
0
0
10
20
30
40
50
60
70
Threads



Read speeds
Milvus DB
Chroma DB
Weavite
Redis
Time (ms)
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
Threads



Recall
Milvus DB
Chroma DB
Weavite
Redis
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
Recall
0
10
20
30
40
50
60
70
Threads

Threats to validity of this approach

Even on a perfect system, recall < 1.

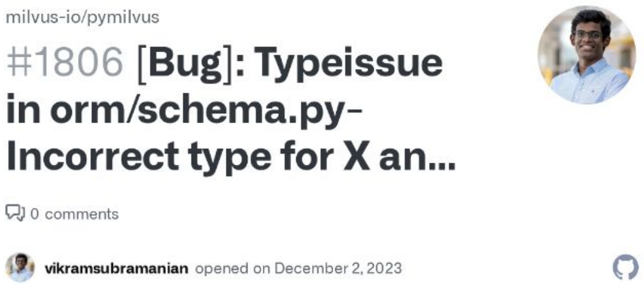
Ie, our data is imperfect

Ie, given a code block x and mutation x’, there exists a code block y whose cosine similarity to x is greater than the cosine similarity between x and x’

Databases under minimal load produced a recall of 0.99 with a perfect dataset. The same DB produced a recall of 0.94.

Conclusion

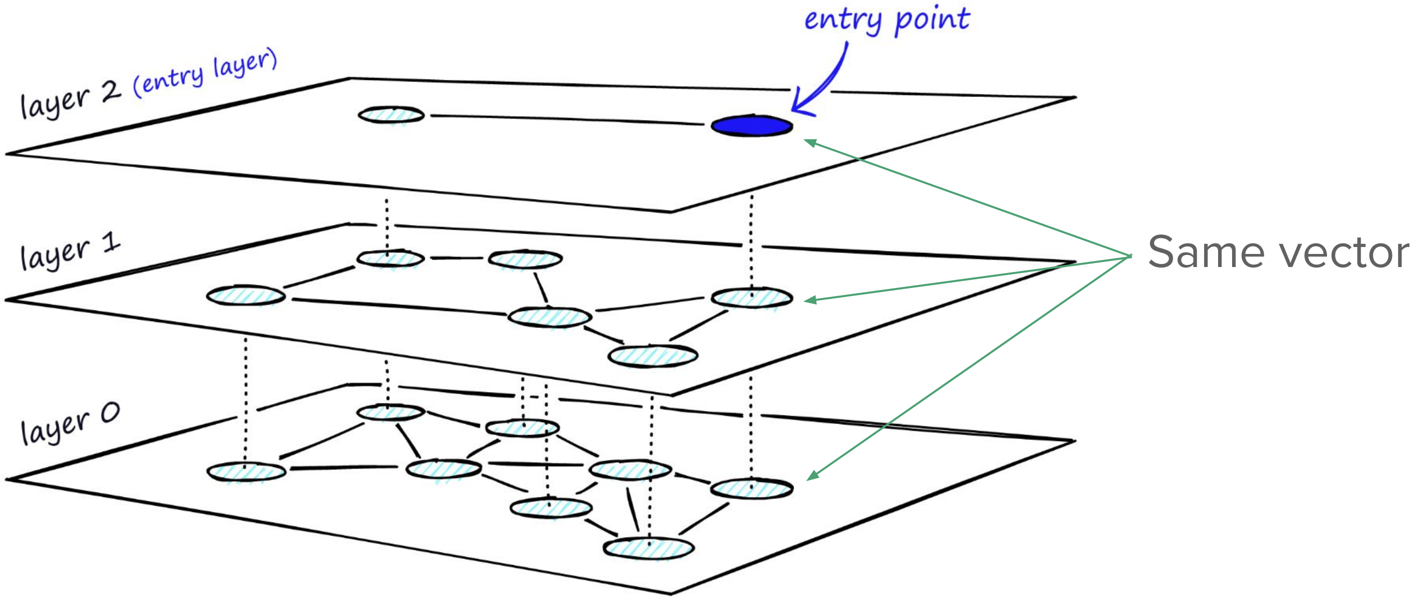
- Created a large high-dimensional dataset for VectorDB benchmarking
- Used a novel method to create a code dataset
- Benchmarked some popular vector databases



Thanks for listening!

How do they work? - Indexing

Hierarchical Navigable Small Worlds



How do they work? - Indexing

Hierarchical Navigable Small Worlds