6.S978 AI ENGINEERING

LECTURE 6

# Retrieval and Reasoning over Massive Contexts

Omar Khattab

Engineering AI Systems and Agents

# In Lecture 4, we decided to automate the how.

**Principles**
stays

**Lecture 2** Requirements | **Lecture 3** Declarative programming | **Lecture 7** Evaluation

Earlier in the semester, we were AI engineers, serving stakeholders.

**Programming systems that learn**
absorbs the change

| Learning algorithms, or compilers | | Inference algorithms, or harnesses | |
|---|---|---|---|
| **Lecture 8** Downstream Learning for AI Systems | **Lecture 9** Downstream Learning in Weights | **Lecture 4** Agent Design | **Lecture 6** Retrieval and Reasoning over Massive Contexts **today** |

Now we are AI systems researchers, serving AI engineers.

**Models**
improves fast

**Lecture 1** Language models

Today we map the landscape, not every detail. You choose where to dive deeper.

Engineers say **declaratively** what their stakeholders need. Our systems make the most of **today's models** to meet it.

We asked: can we find a principled approach to engineering **intelligent systems** that are reliable, efficient, and maintainable?

So far, **reliability** and **maintainability**. Today, **efficiency**.

# Today

1. Massive contexts in practice
2. Neural information retrieval
3. Beyond exhaustive search
4. Bridging the gap to brute force

Part 1

# Massive contexts in practice

payments/retry.py

```
def charge(order):
    for i in range(3):
        try: return pay(order)
        except Timeout: wait(i)
```

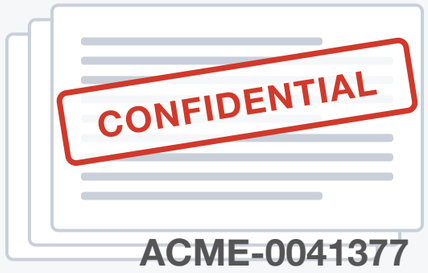
**A codebase**



**The research literature**

| | | |
|---|---|---|
| #48213 | **Checkout fails** | Open |
| #48207 | **Can’t log in** | Closed |
| #48199 | **Crash on launch** | Open |
| #48186 | **Refund missing** | Pending |

**Past support tickets**




**A lawsuit’s documents**

| | |
|---|---|
| **ITA Airways** | Updated itinerary |
| **Shopify** | Order #1047 |
| **Priya Raman** | Re: Q4 plan |
| **Lena Ortiz** | Re: mugs |

**An inbox**

## Inbox 100,000 emails since 2020

| | |
|---|---|
| **ITA Airways** | **Updated itinerary: Boston → Rome, Oct 22** |
| **Shopify** | **[Juniper Pottery] Order #1047** |
| **Priya Raman** | Re: Q4 plan — numbers by Friday? |
| **Lena Ortiz** | Re: mugs for our café |
| **Mark Chen** | Re: order #1032 — any news on the replacement? |
| **Northgate Properties** | Lease renewal |

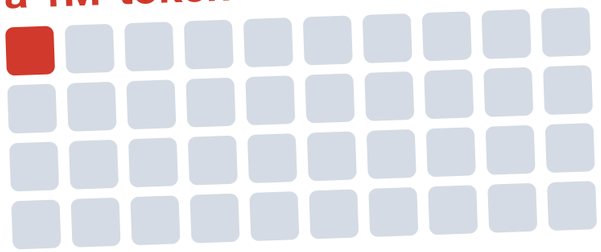
✦ **Ask your inbox**

Ask anything

| | |
|---|---|
| **Simple** | When does my flight to Rome leave? |
| **Multi-hop** | Did the café that asked about wholesale make an order? |
| **Aggregation** | What did the shop earn this year? |
| **Combinatorial** | Did anyone order the same piece twice? |
| **Oblique** | Which customers have a major unresolved problem with their order? |
| **Familiarity** | Which of this week's orders look unusual? |

**How could a system answer them?**

a 1M-token window

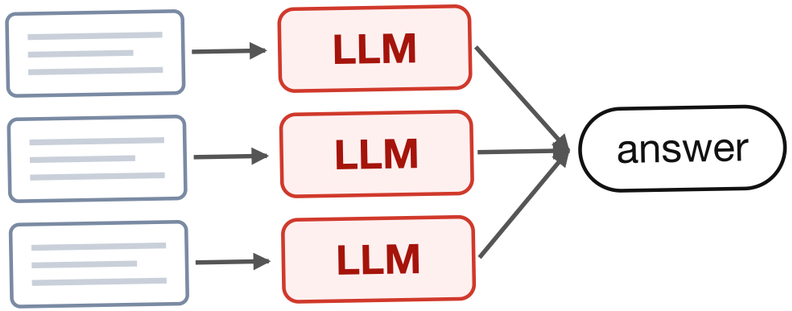


the inbox: 40M tokens

**Put the whole inbox in the prompt**

```
$ grep -il rome inbox/*
31877.eml  40122.eml
48210.eml  52301.eml …
```

**Let an agent search it**




**Split it, and have a model read each part**

```python
class AskInbox(Signature):
    """Answer my question about my inbox."""
    inbox: list[Email] = InputField()
    question: str = InputField()
    answer: str = OutputField()
```

**Is this quite the right signature?**

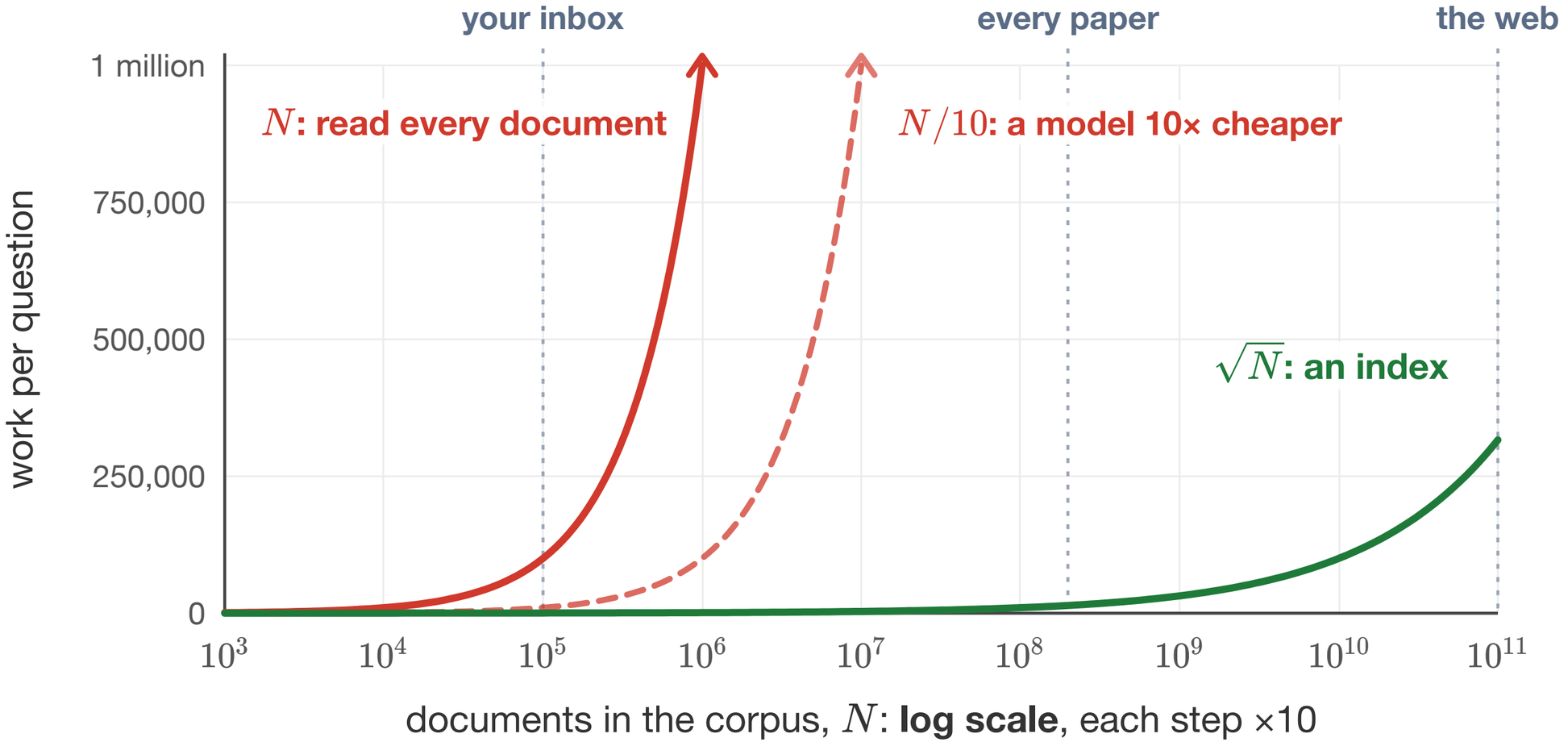
## At indexing time: once per email

```
class IndexInbox(Signature):
    inbox: list[Email] = InputField()
    index: Index = OutputField()
```

## At query time: every question

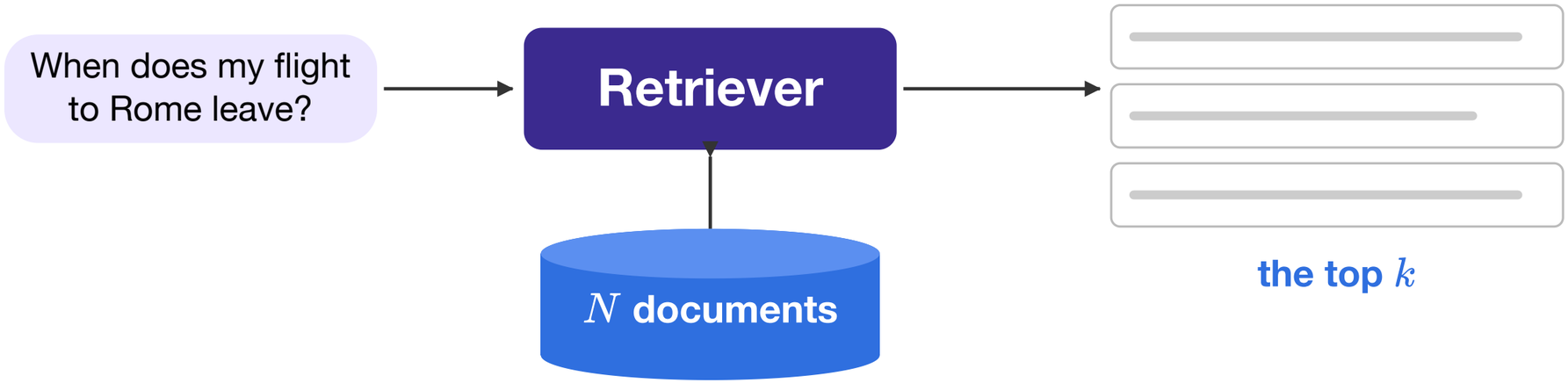
```
class AnswerFromIndex(Signature):
    index: Index = InputField()
    question: str = InputField()
    answer: str = OutputField()
```

# Maybe just wait for a **better model**?




**So what should we compute at indexing time?**

# Information retrieval at a glance




```python
class Retrieve(Signature):
    """Find the documents that answer the question."""
    question: str = InputField()
    index: Index = InputField()
    top_k: list[DocID] = OutputField()
```

**Quality**

as good as reading
every document
with the question

**Latency**

milliseconds,
as $N$ grows

**What would you compile this signature to?**

Part 2

# Neural information retrieval

# End-to-end retrieval with an LLM

**Prompt**

```
Email 1: Booking confirmed: Boston → Rome, Oct 21…
Email 2: [Juniper Pottery] Order #1001 from…
⋮
Email 100,000: Updated itinerary: Boston → Rome…

Question: When does my flight to Rome leave?
Return the IDs of the 10 best emails.
```

→ `100000, 1, 40122, …`

**What's good about this, and what's bad?**

**How might you improve it?**

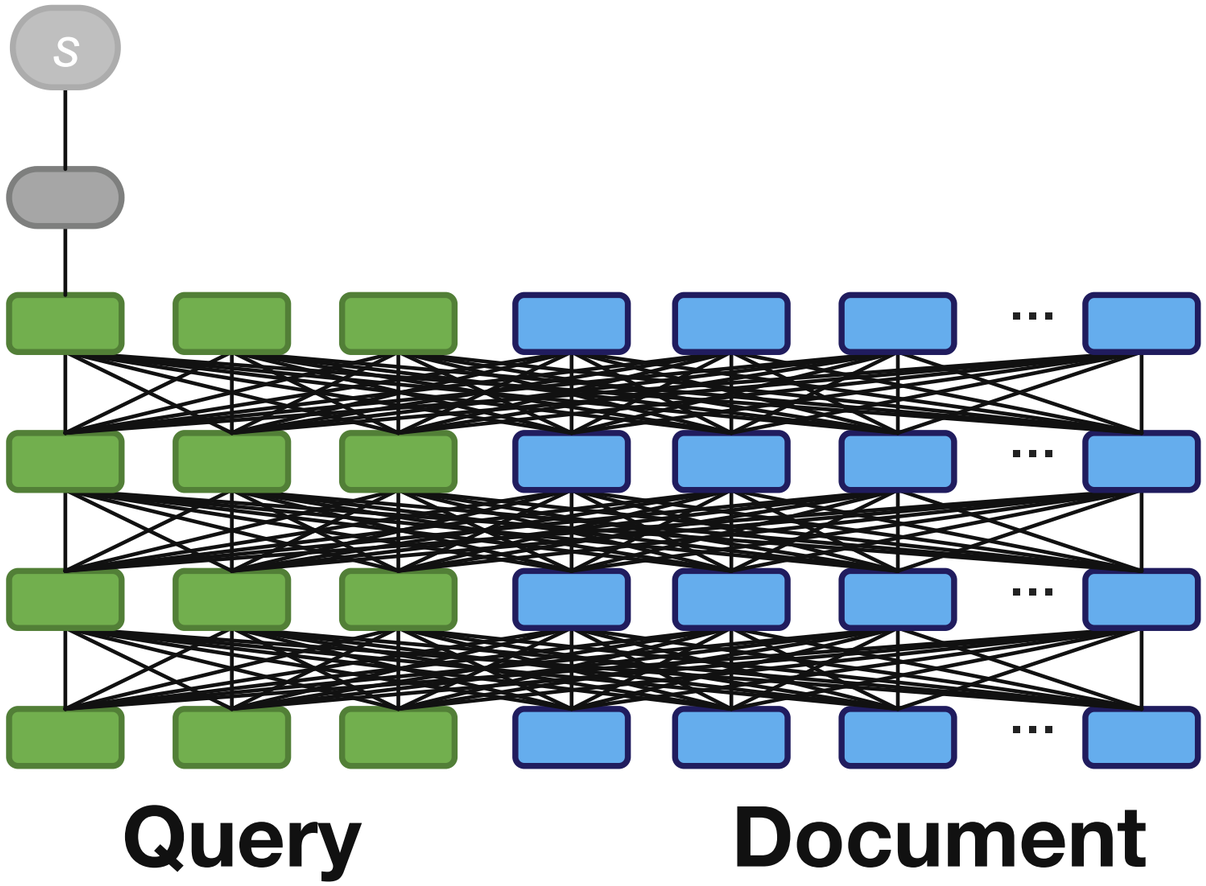
# Decomposition: one LLM call per email

```python
class Relevance(Signature):
    """Rate the email's relevance to the question, 0 to 10."""
    question: str = InputField()
    email: Email = InputField()
    score: int = OutputField()
```

→ 9

Assumption: each email can be scored on its own. Is this true?

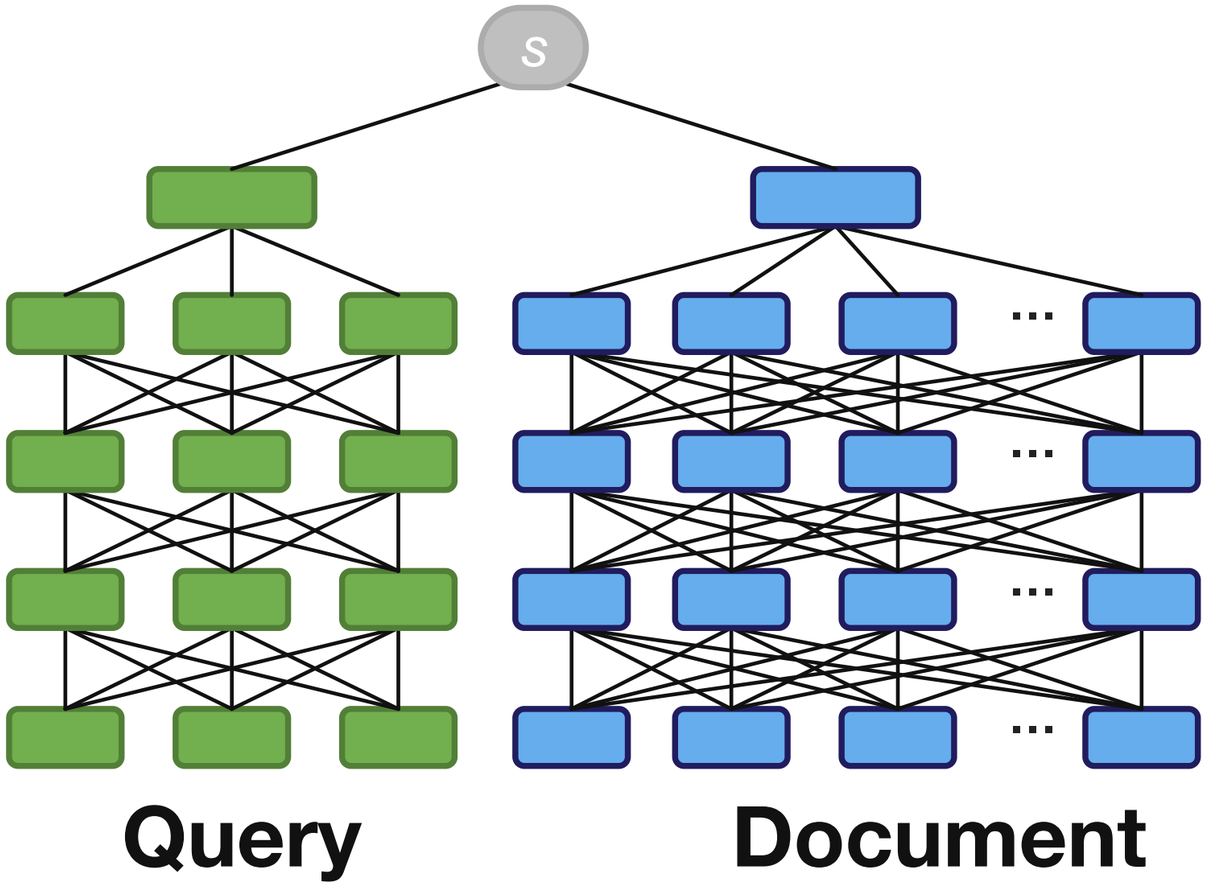
# Cross-encoders: a Transformer trained to score each pair




Per question:
$\mathcal{O}(N)$ Transformer calls

Can we do better?

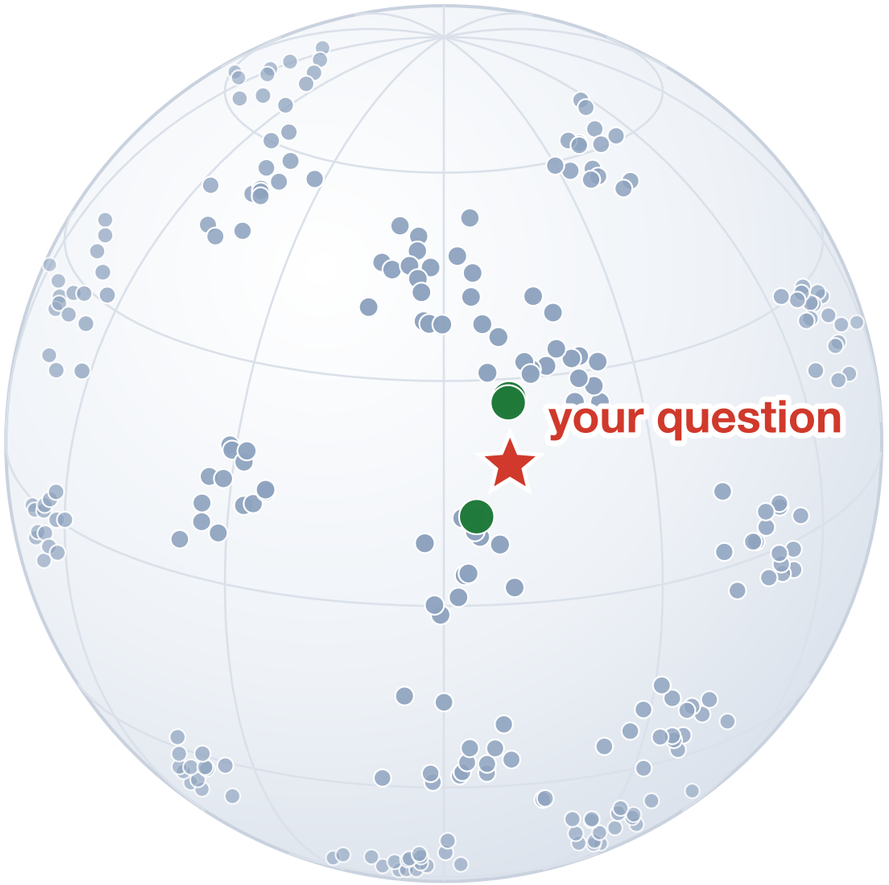
# Dual encoders




$$s(q, e) = E_Q(q)^\top E_E(e)$$

Indexing, once:
$\mathcal{O}(N)$ Transformer calls

Per question:
$\mathcal{O}(N)$ dot products

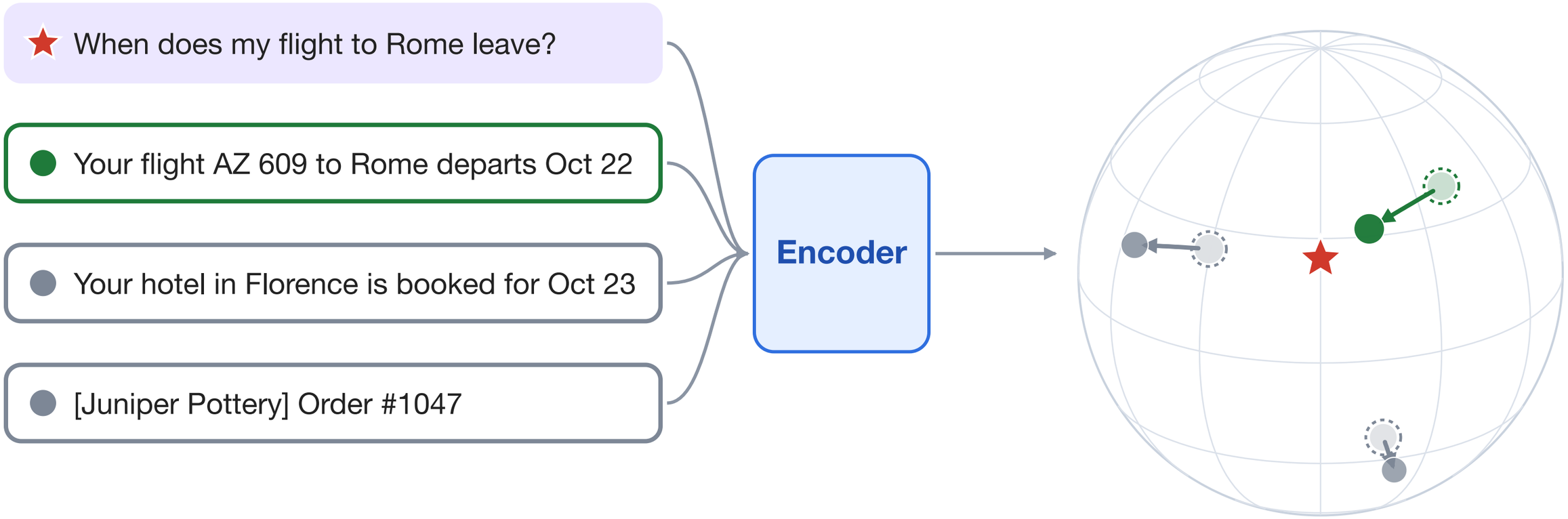



3 of 768 dimensions

Assumption: an email can be a single vector that doesn't depend on the question. Is this true?

**How would you train this?**

# Contrastive learning



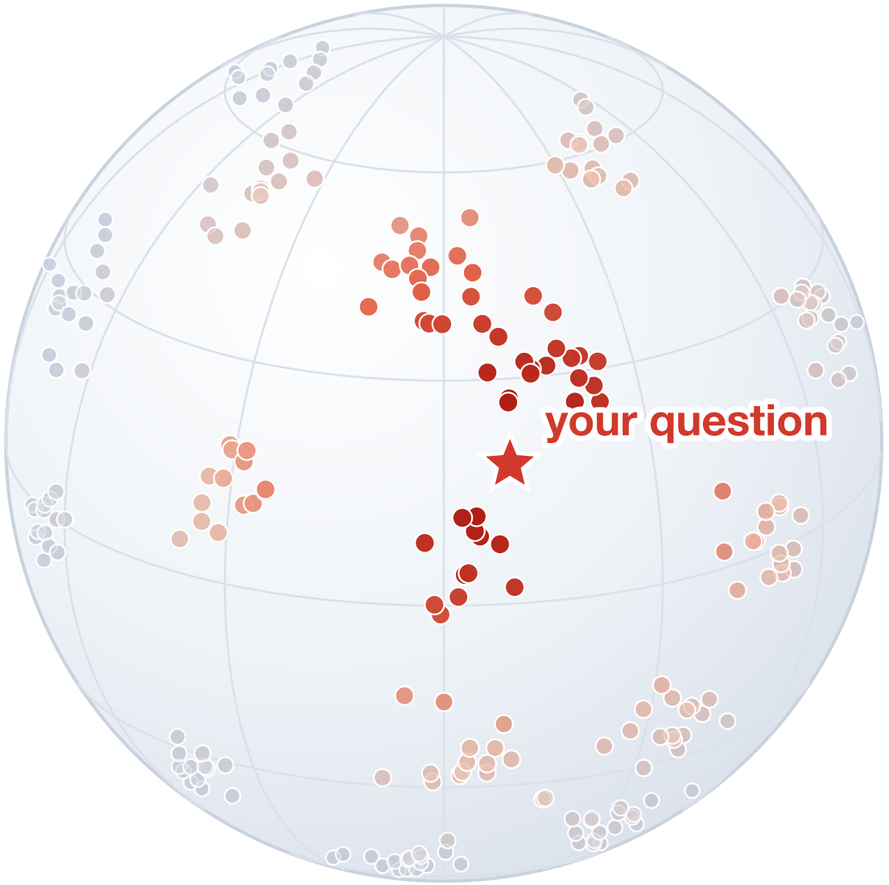

Pull the positive in. Push the negatives away.

**Where do the questions and the negatives come from?**

Part 3

# Beyond exhaustive search

# Are (even) dual encoders fast enough?

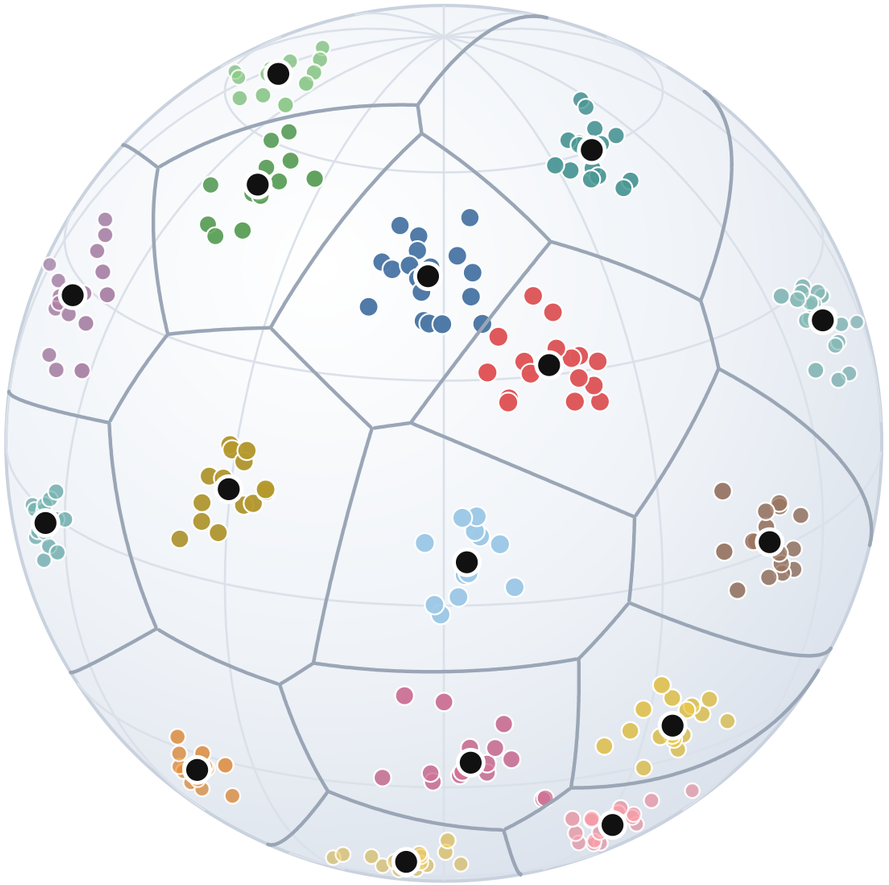



So far, one Transformer call per question.

But still $N$ dot products: 100,000.

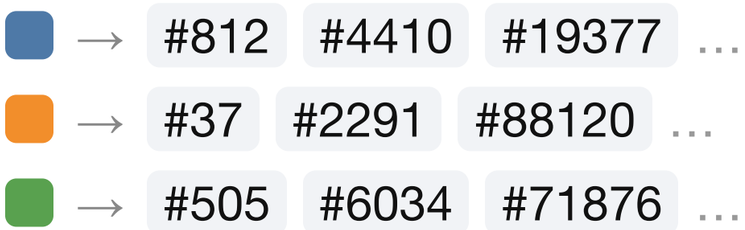
**Can we do (much) faster?**

# A simple heuristic: the inverted file (IVF)



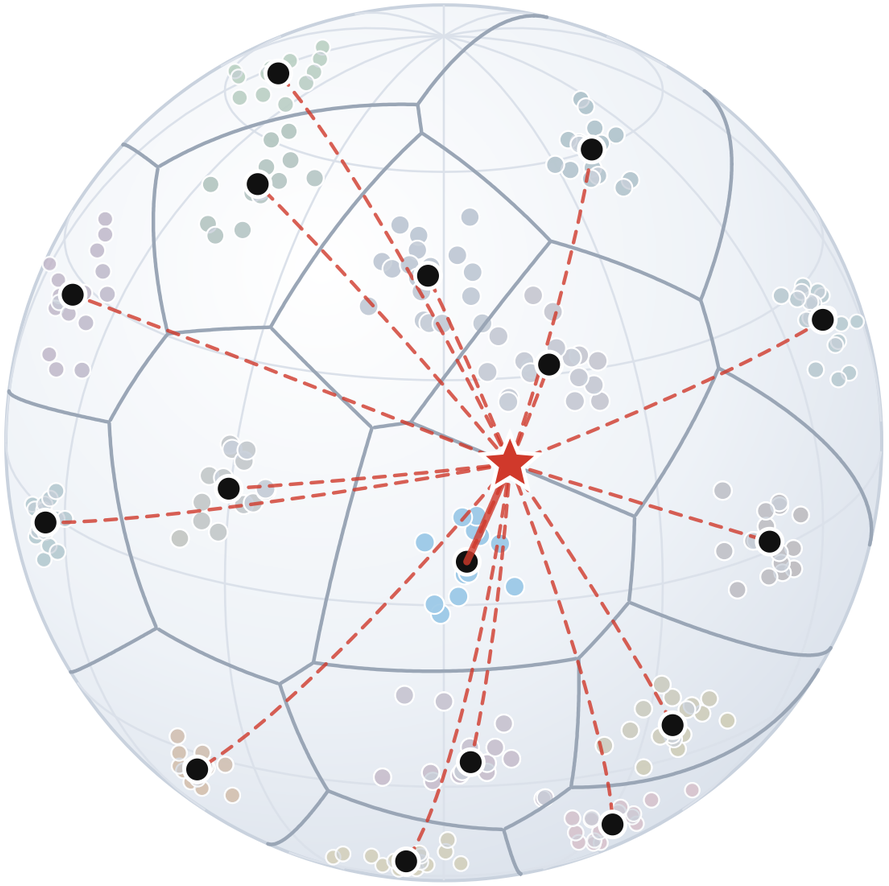
At indexing time, cluster the $N$ vectors into $K$ groups.

Store an inverted list per group.




Assume roughly equal-sized groups: $N/K$ each.

# At query time



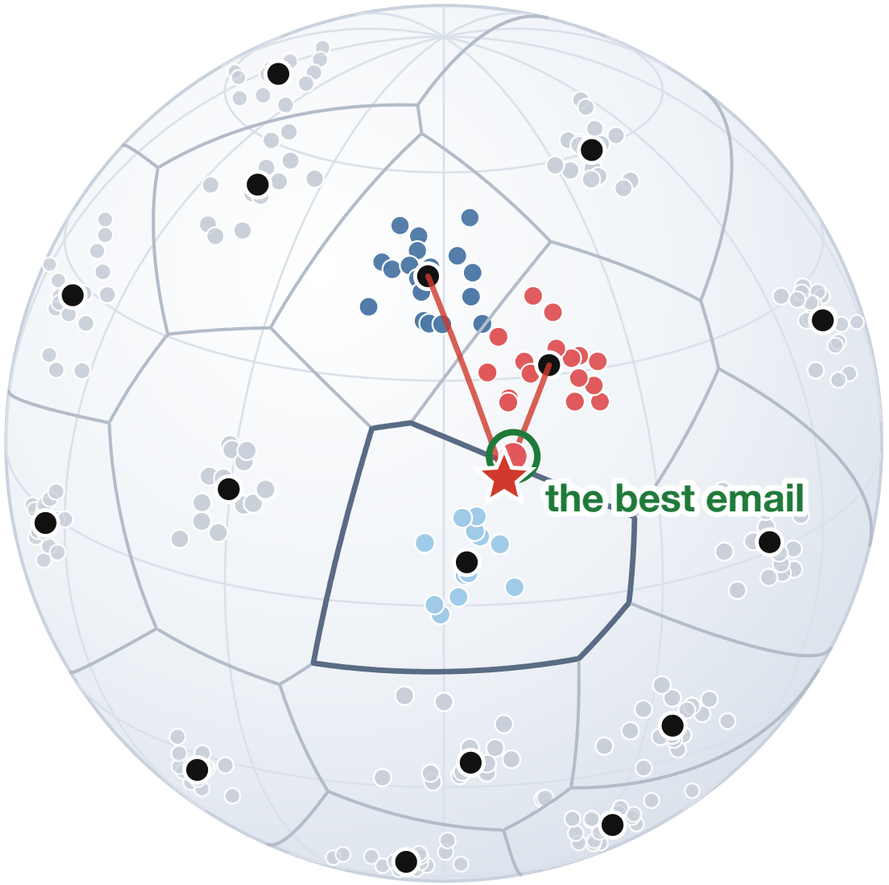
Compare the question with all $K$ centers. Pick the closest.

Only scan that list: $N/K$ more dot products.

$K + N/K$ is smallest at $K = \sqrt{N}$.
About $2\sqrt{N}$: 632, not 100,000.

**What did we assume?**

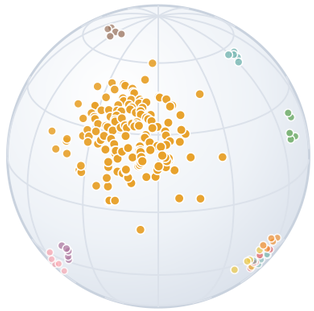
# But is it high-quality enough?




We assumed the best email is in the closest group. It can sit just across a border.

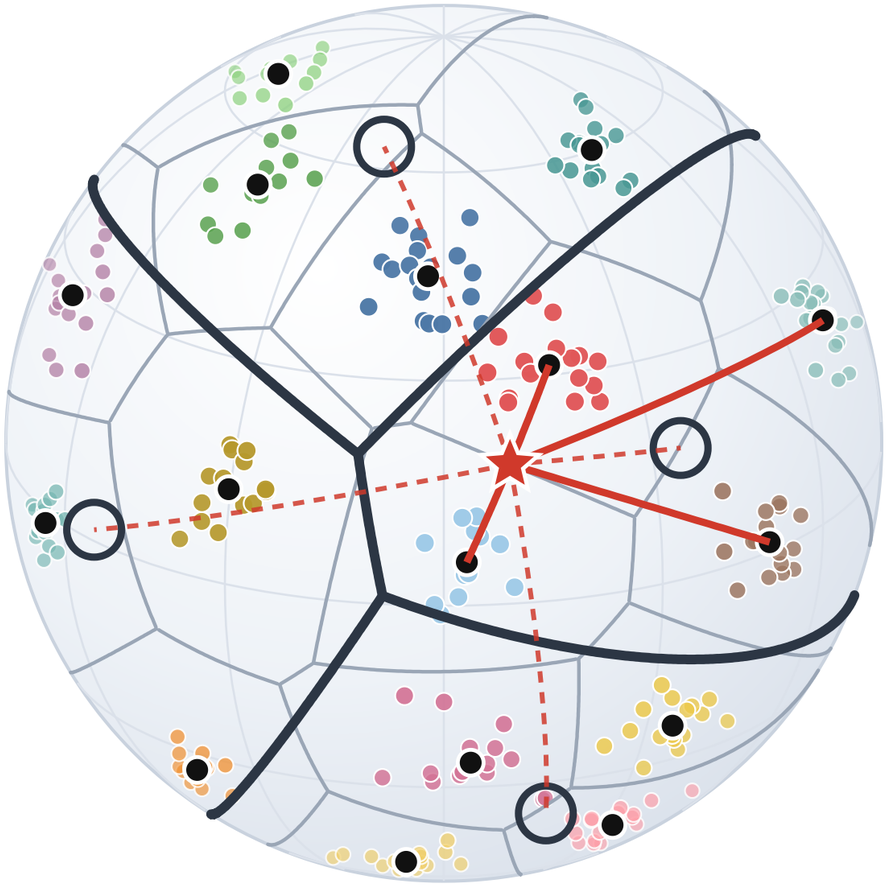
So probe the nearest few groups.

In 768 dimensions, a group can border all 315 others.



We assumed equal groups. They come out uneven.

# Groups of groups



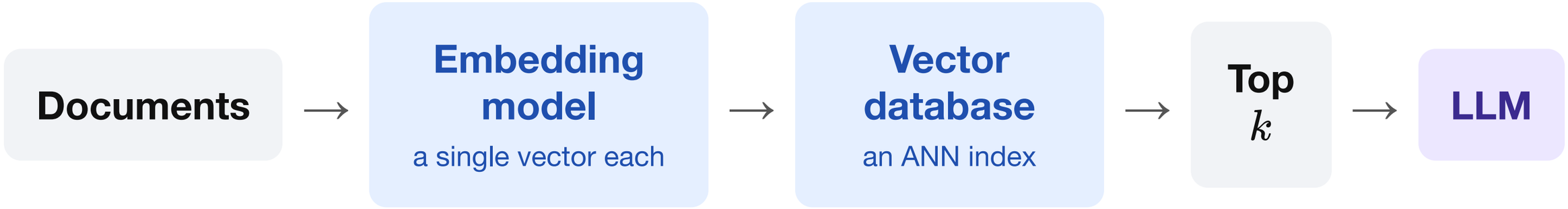
The $\sqrt{N}$ centroids are vectors too.

So cluster them: $\sqrt{\sqrt{N}}$ groups of centroids.

Recursively: a tree, $\log N$ levels deep.

At each level, probe a few groups.

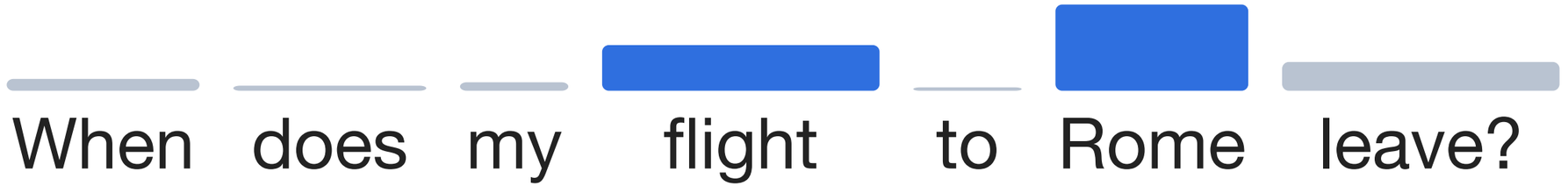
# Most retrieval today works this way

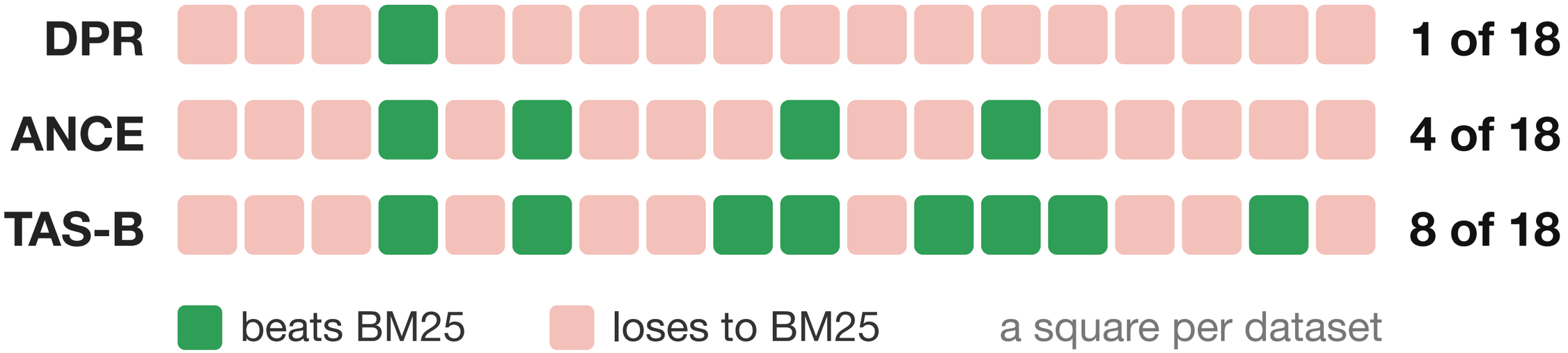


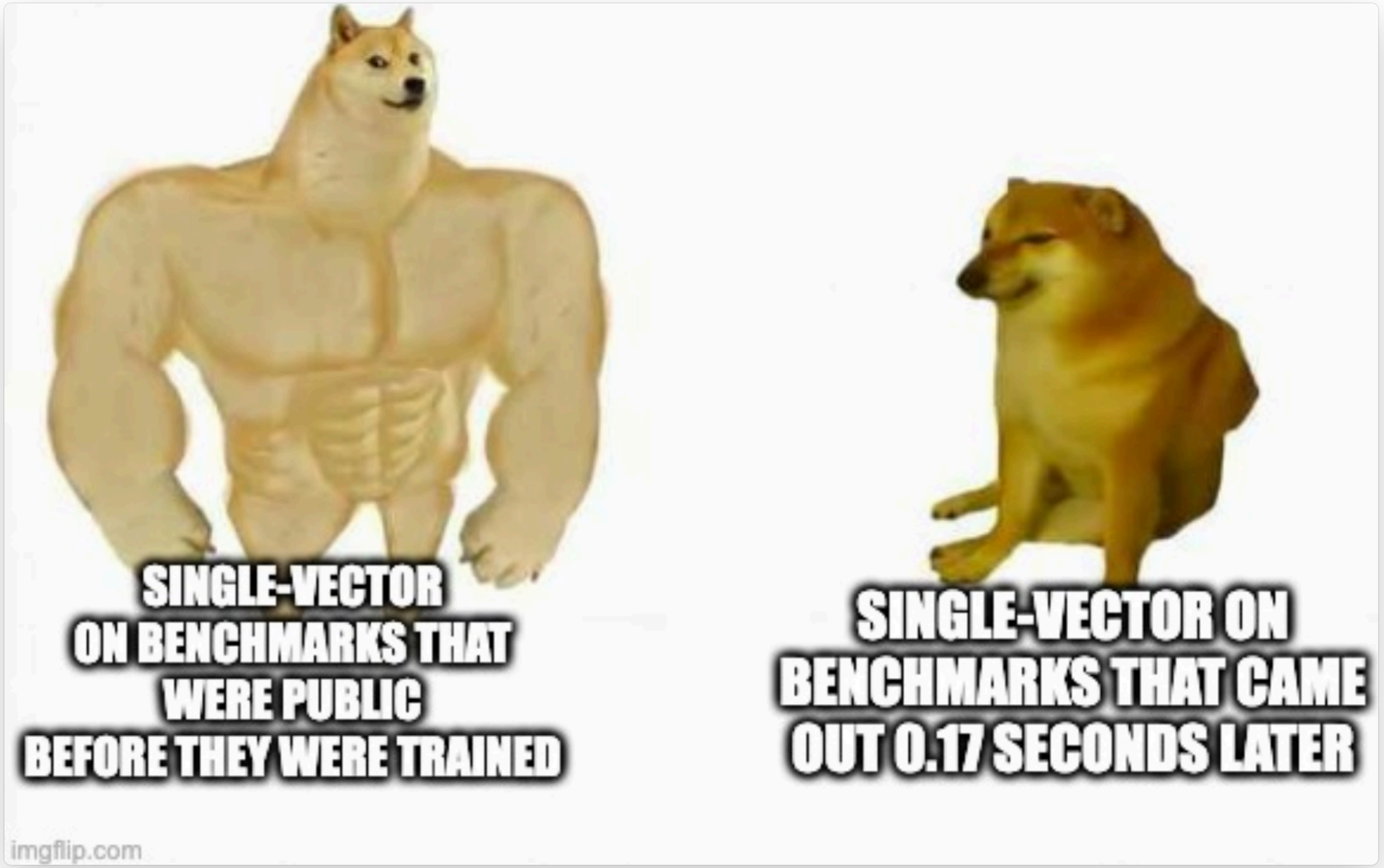

The default in RAG tutorials, vector databases and embedding APIs.

**Even with exact search, would the results be right?**

BM25, from 1994: matching words, weighted by how rare they are.






Meme: Ben Clavié.

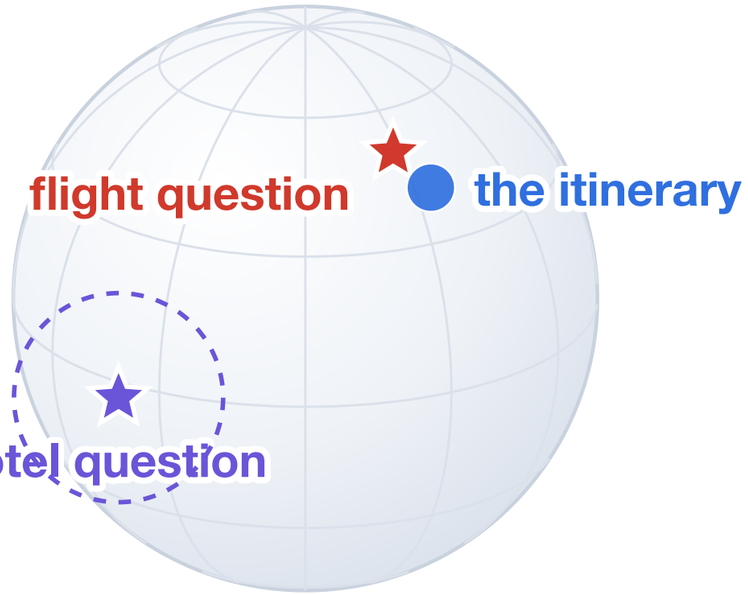
# Why would a single vector fall short?

Updated itinerary: Flight AZ 609 to Rome now departs Oct 22 at 9:40 pm.
Your hotel in Florence is booked for Oct 23.

**Should its vector move?**




If it moves, an ad outranks it.



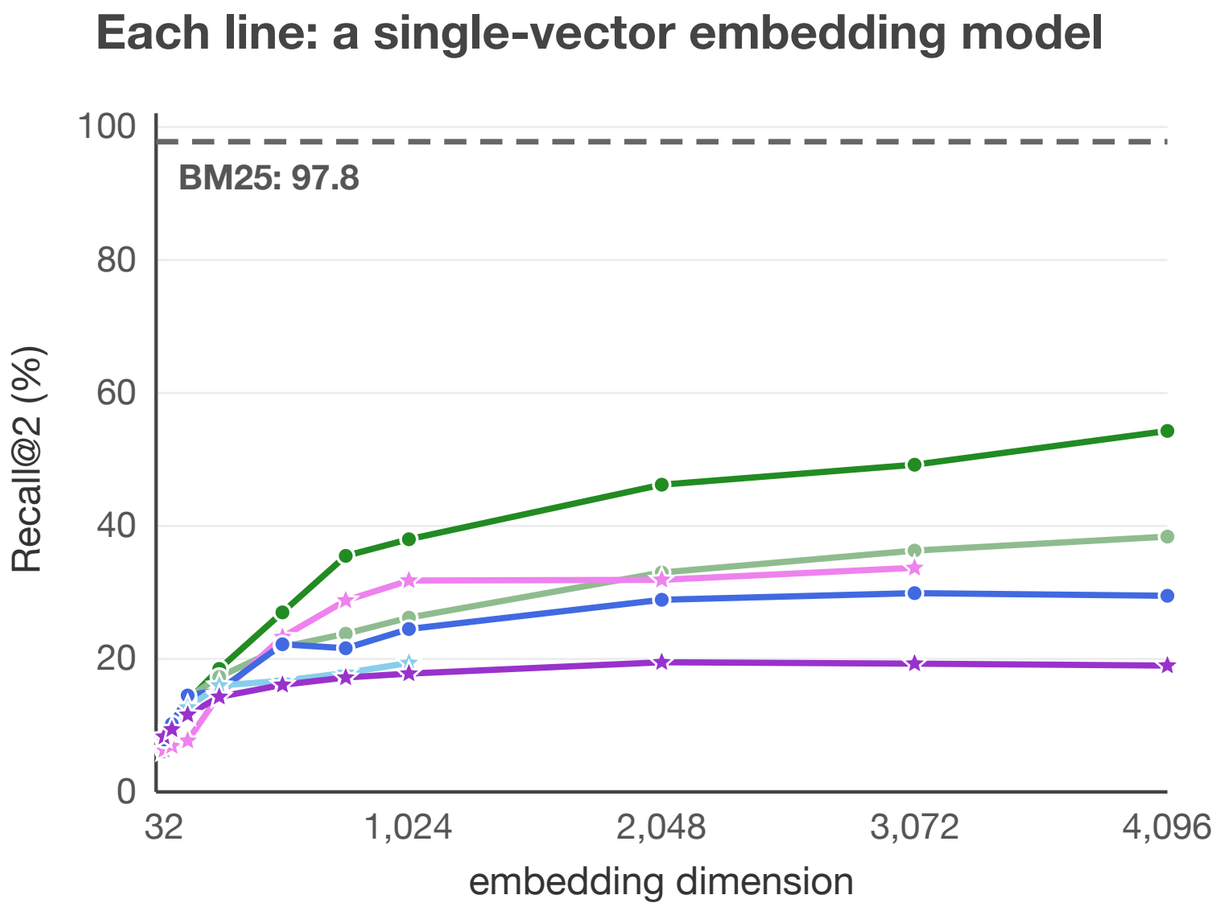

If it stays, the hotel is invisible.

Jon Durben likes quokkas and apples.

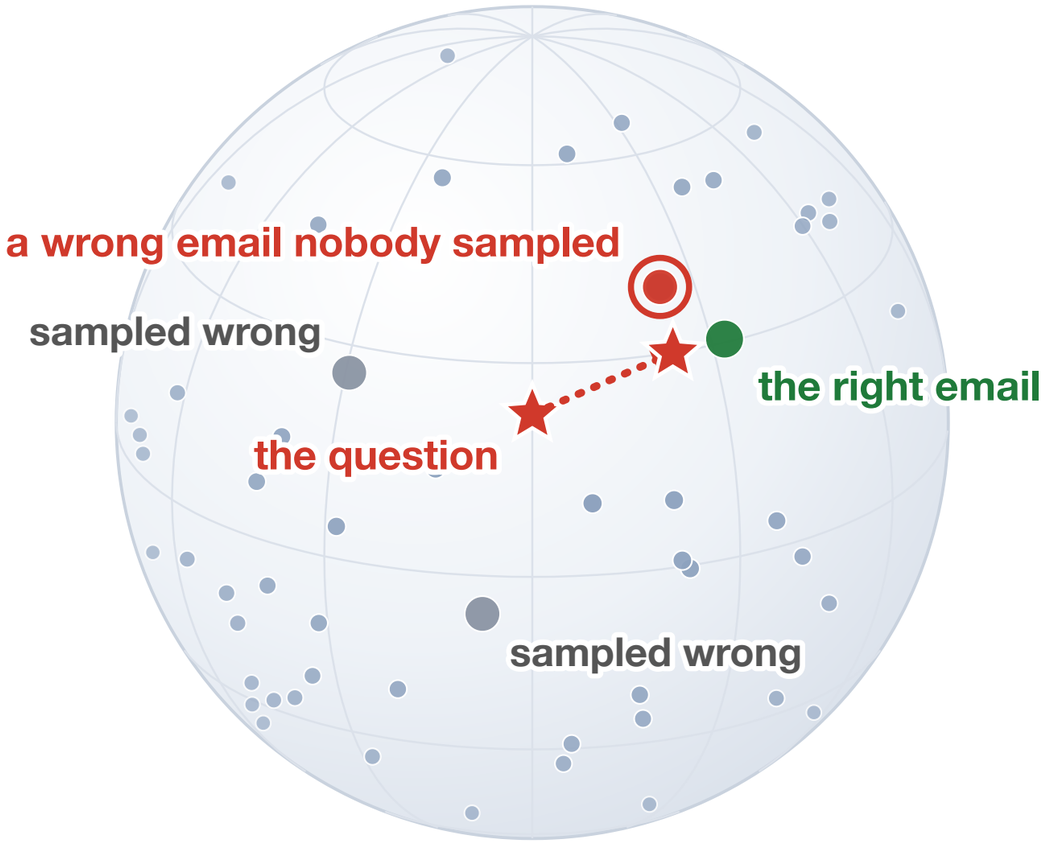
Ovid Rahm likes quokkas and rabbits.

Leslie Laham likes apples and candy.

… 46 people in all



# In practice, it's often a **generalization** issue

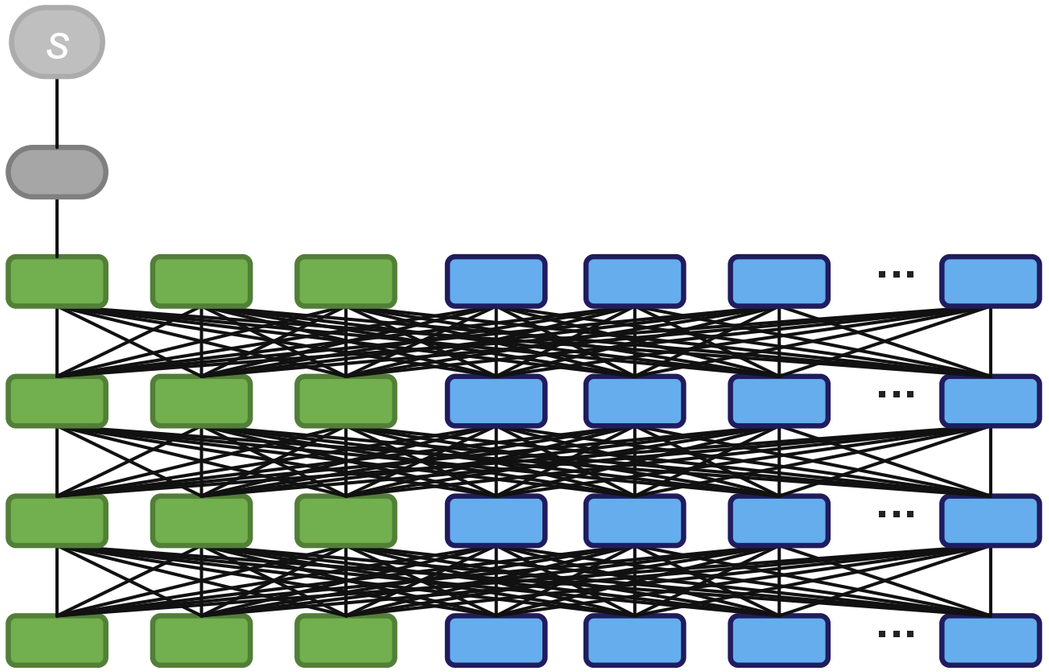



Blunt representation,
blunt updates.

Push the question's vector
from a wrong email,

and it moves toward others!

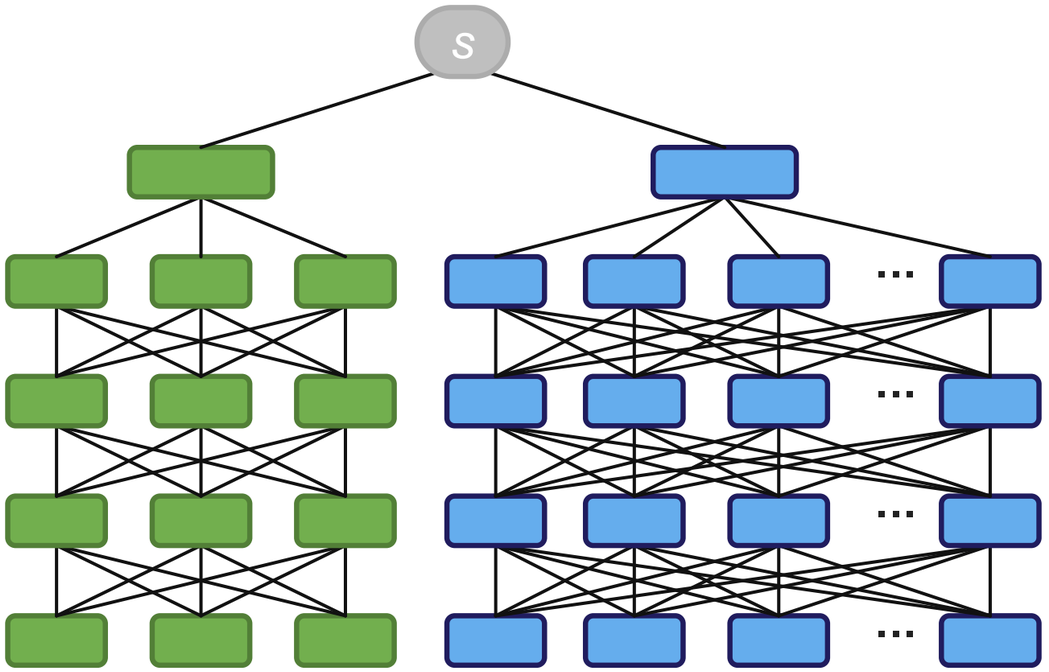
**Can we keep this efficiency, but do much better on quality?**




**Cross-encoder**

✓ High quality

✗ Unscalable



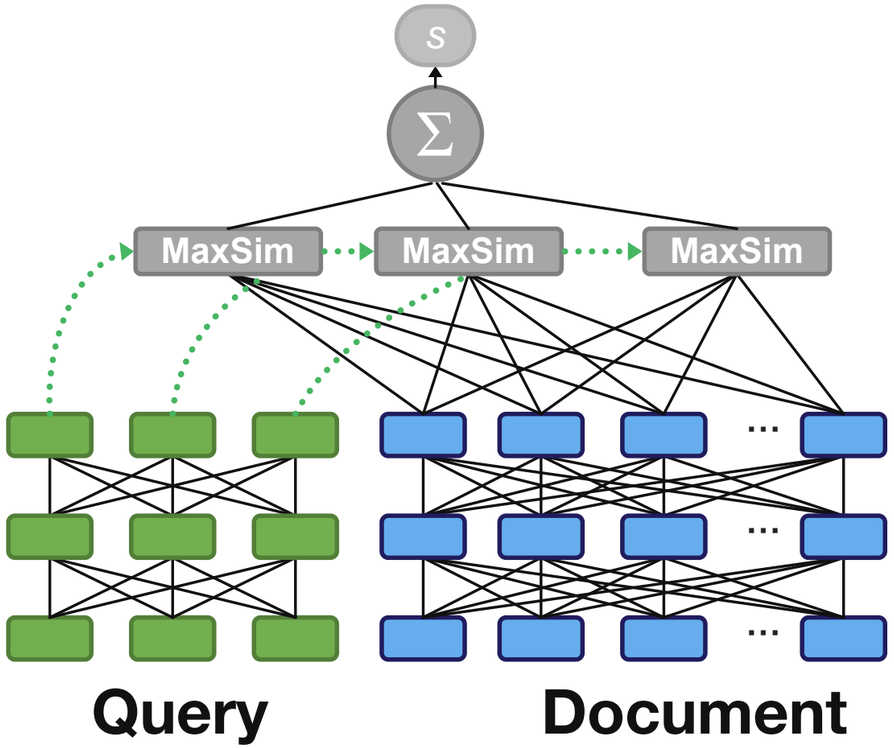

**Single-vector dual encoder**

✓ Highly scalable

✗ Low quality

# ColBERT: the **late interaction** paradigm

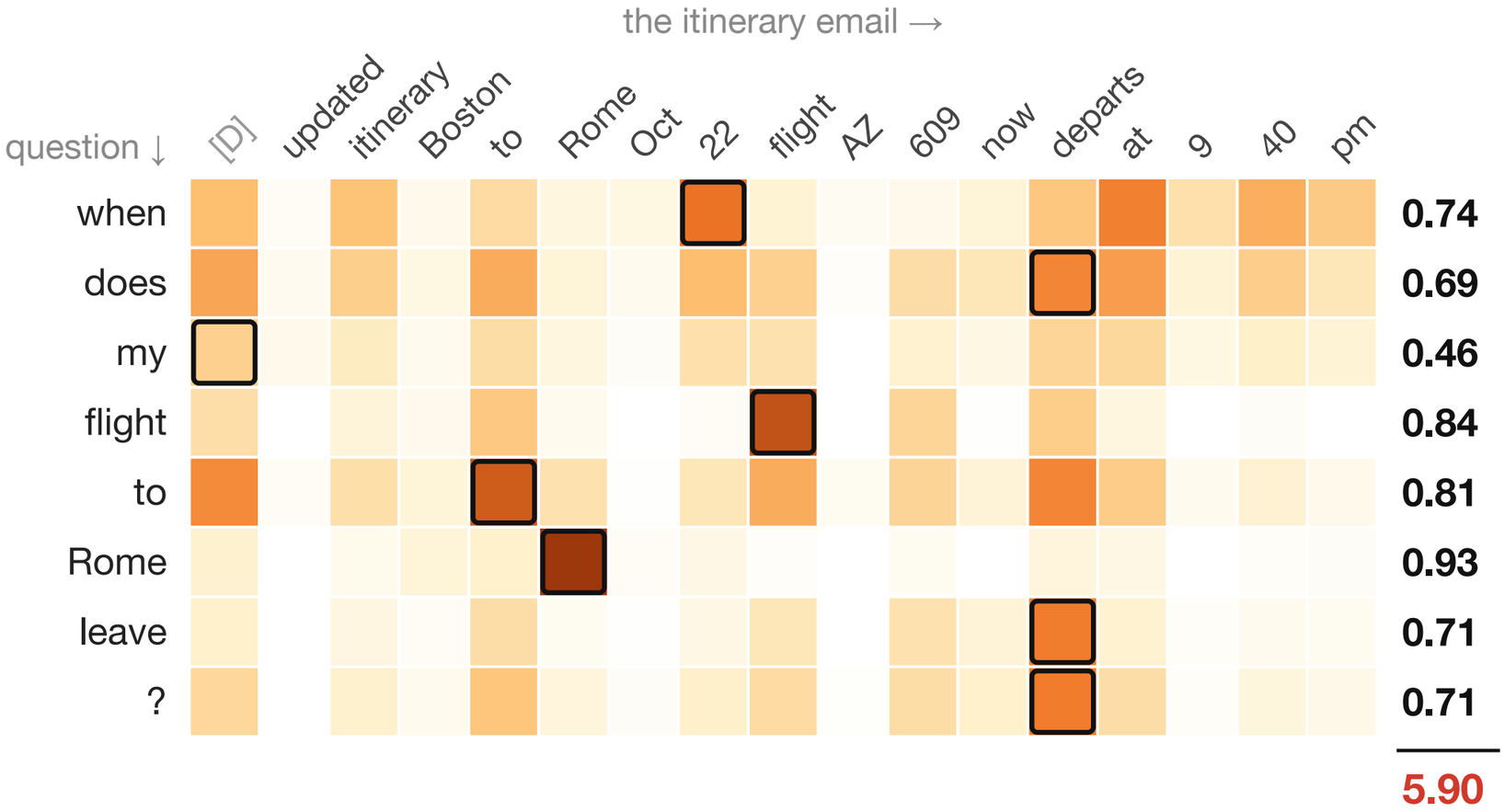
**Hypothesis:** the main value of attention for IR is the fine-grained interactions. **Jointly** encoding the question and the document is **redundant**.




- ✓ **Independent computations**
- ✓ **Fine-grained representation**
- ✓ **Attention-like top-k search** ***in*** $O(\sqrt{N})$ ***time***

$$S_{q,e} = \sum_i \max_j E_{q_i} \cdot E_{e_j}^\top$$

# Q: When does my flight to Rome leave?



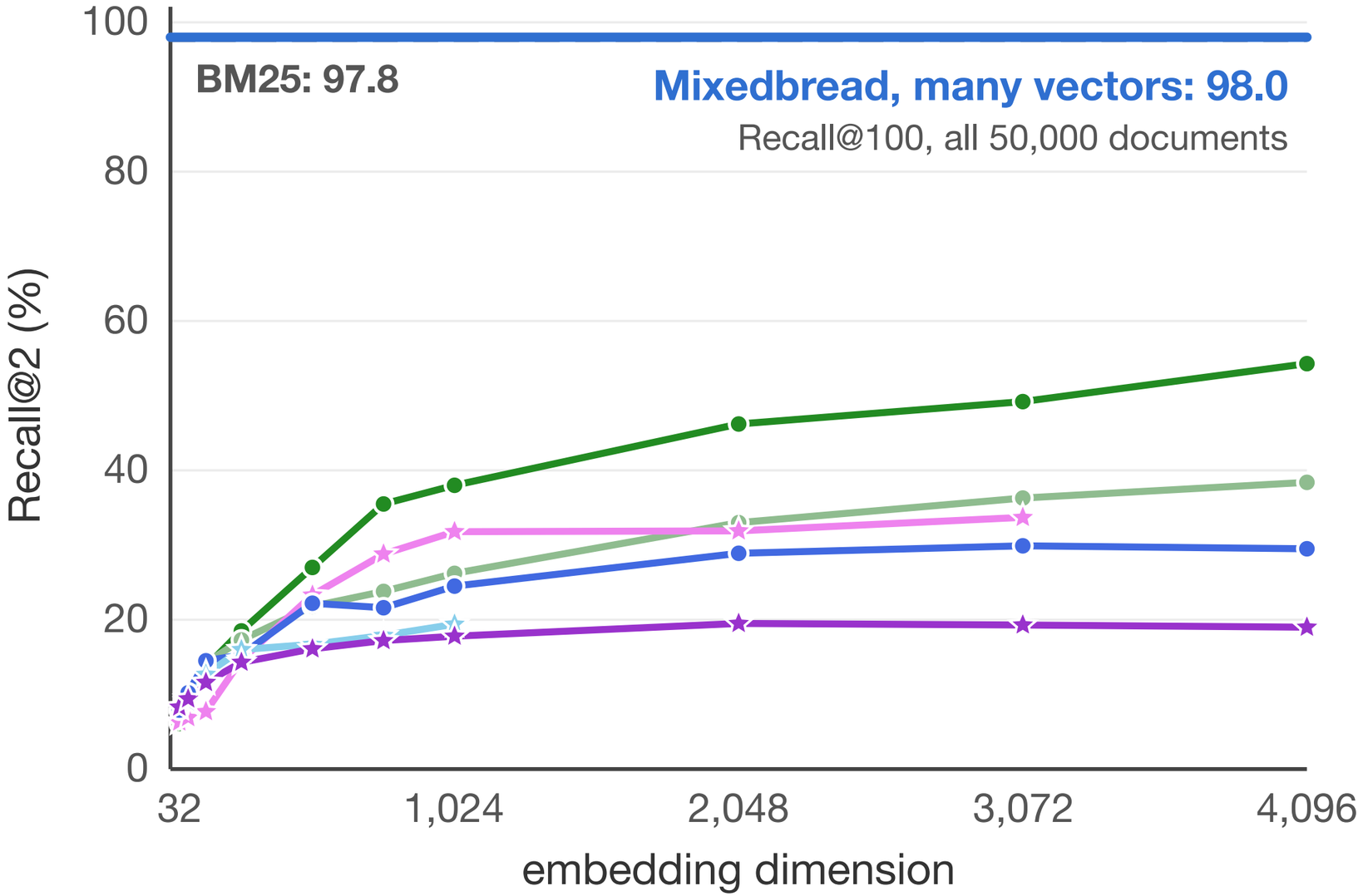

**Ranked by score**

| | |
|---|---|
| Booking confirmed: Boston → Rome, Oct 21 | **5.96** |
| **Updated itinerary: Boston → Rome, Oct 22** | **5.90** |
| Hotel Artemide: your stay in Rome | **4.46** |
| Your 2025 trip to Rome: tell us how we did | **3.40** |
| Your Wednesday morning trip with Uber | **2.21** |

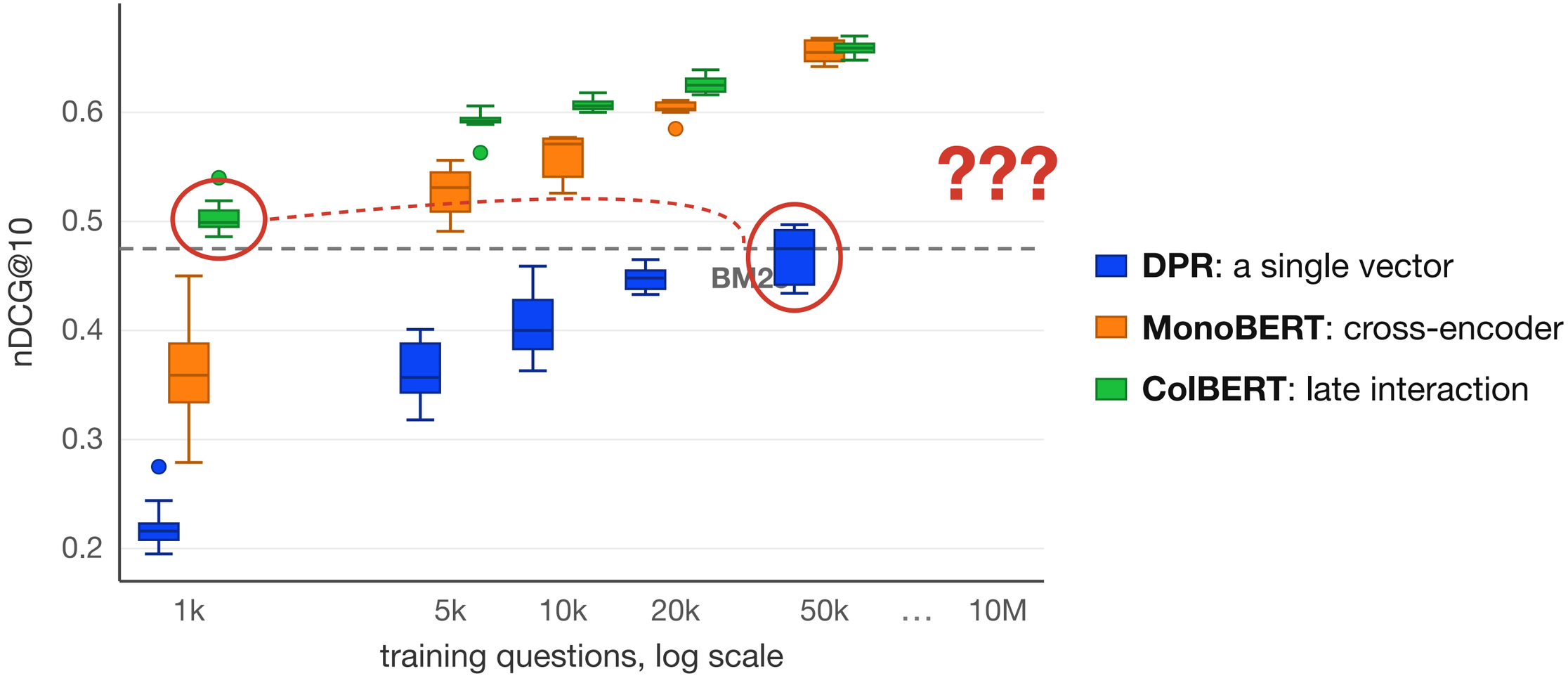


Similarities from the ColBERTv2 checkpoint. Santhanam, Khattab, et al., “ColBERTv2,” NAACL 2022.

## Each line: a single-vector embedding model



# What are the effects of these **assumptions**?



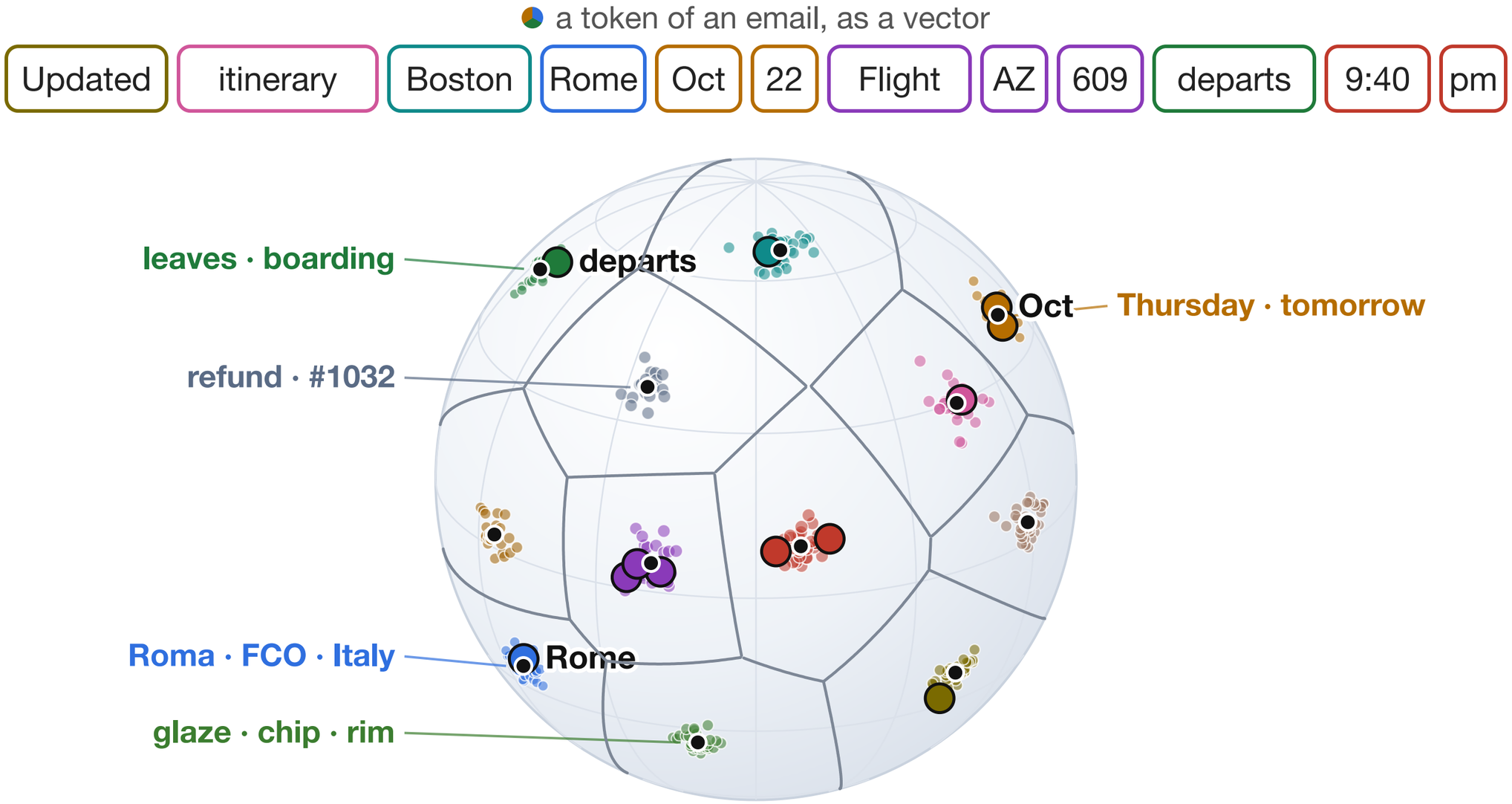
The immediate challenge is
hundreds of vectors per email, instead of one.

**1** Storage blowup?

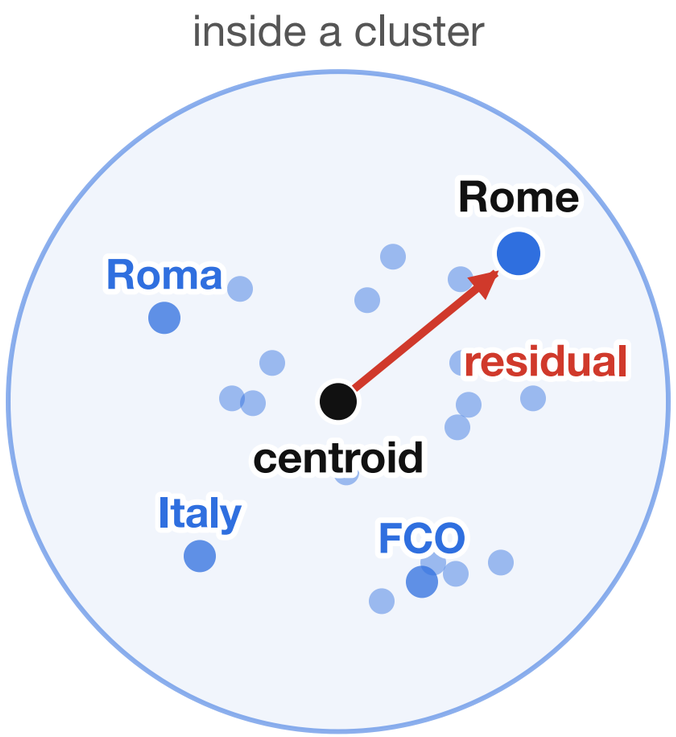
**2** Latency blowup? How do we search
in about $\sqrt{N}$ when an email is a matrix?

**Let's tackle both!**

# Vectors capture contextualized words



# Vector = centroid ID + residual

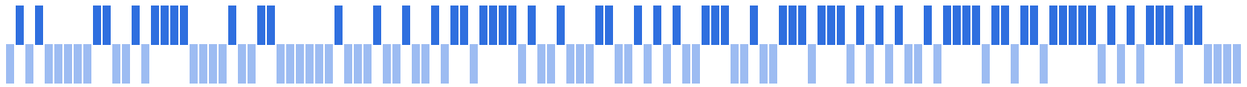



an email token  a centroid

the residual: 128 numbers, all small



keep 1 bit of each: up or down



a raw vector: 128 numbers, 16 bits each

**256 bytes**

centroid ID, 4 bytes, plus 128 bits, 16 bytes

**20 bytes**

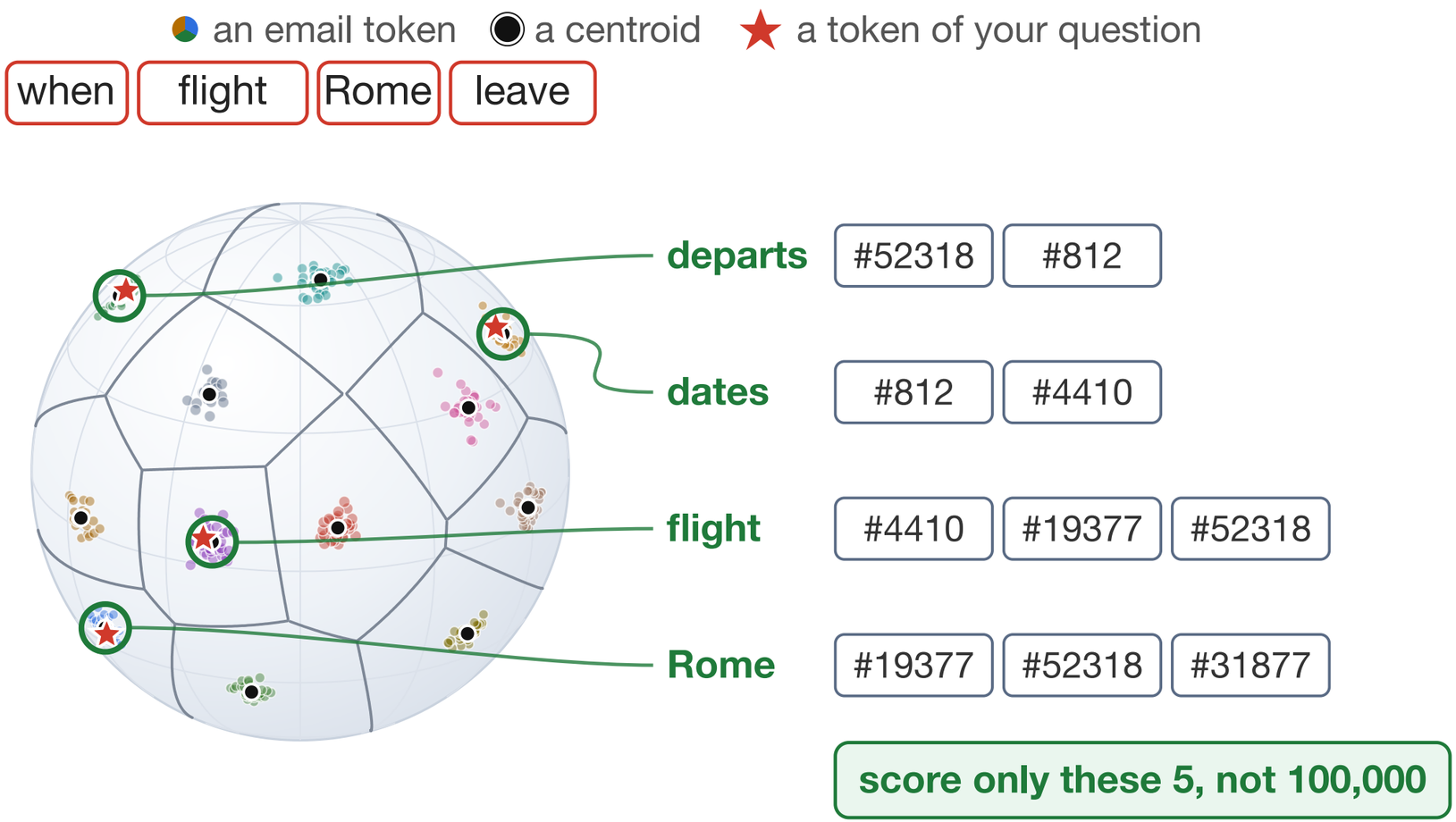
an email: 400 vectors × 20 bytes

**8 KB**

a single vector: 4,096 numbers × 16 bits

**8 KB**

# Only score emails with a token near the question




Heuristic: a high score needs a close pair of vectors.

About $\sqrt{N}$ dot products. Today, about 10 ms on a CPU core.

# Scaling is necessary, but can't always make up for poor assumptions!

Table 3: **Agent performance across retrievers on BrowseComp-Plus. The agent is based on gpt-oss-20b; it supports a maximum context window of 131,072 tokens. "#Search" and "#GetDoc" denote the number of search and full-document reader calls, respectively. "Compl." denotes the completion rate, i.e., the percentage of queries for which the agent outputs a final answer before reaching the maximum context-window or output-token limits; queries reaching these limits receive accuracy 0. The best values in recall and accuracy are boldfaced, and the second-best ones are underlined.**

| Retriever | Passage corpus | | | | Document corpus | | | | Document corpus+GetDoc | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | #Search | Recall | Acc. | Comp. | #Search | Recall | Acc. | Comp. | #Search | #GetDoc | Recall | Acc. | Comp. |
| BM25 | 30.97 | **0.616** | **0.572** | 0.968 | 32.22 | 0.366 | 0.259 | 0.805 | 28.14 | 1.31 | 0.343 | 0.301 | 0.746 |
| SPLADE-v3 | 32.75 | 0.545 | 0.516 | 0.980 | 28.95 | 0.628 | 0.476 | 0.824 | 24.64 | 1.65 | 0.602 | 0.529 | 0.829 |
| RepLLaMA | 36.13 | 0.449 | 0.406 | 0.961 | 30.69 | 0.514 | 0.363 | 0.786 | 26.86 | 1.46 | 0.476 | 0.399 | 0.816 |
| Qwen3-Embed-8B | 37.83 | 0.470 | 0.417 | 0.963 | 30.14 | 0.570 | 0.421 | 0.821 | 26.56 | 1.63 | 0.559 | 0.455 | 0.823 |
| ColBERTv2 | 32.88 | 0.552 | 0.521 | 0.978 | 27.13 | **0.633** | **0.481** | 0.855 | 24.31 | 1.75 | **0.612** | **0.538** | 0.835 |

## Is a bigger encoder enough?

ColBERTv2, at 110M parameters from 2021, finds more for an agent than an 8B embedding model from 2025.

✗ 100× more vectors to store

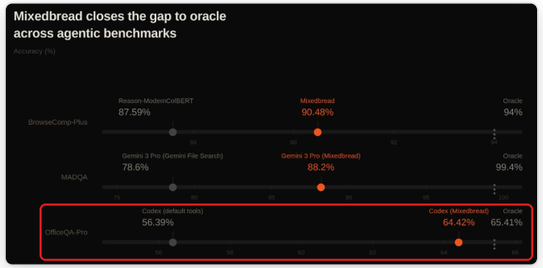
| **Ours** | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SigLIP (Vanilla) | 43.2 | 30.3 | 64.1 | 58.1 | 26.2 | 18.7 | 62.5 | 65.7 | 66.1 | 79.1 | 51.4 |
| BiSigLIP (+fine-tuning) | 58.5 ↑ 15.3 | 32.9 ↑ 2.6 | 70.5 ↑ 6.4 | 62.7 ↑ 4.6 | 30.5 ↑ 4.3 | 26.5 ↑ 7.8 | 74.3 ↑ 11.8 | 73.7 ↑ 8.0 | 74.2 ↑ 8.1 | 82.3 ↑ 3.2 | 58.6 ↑ 7.2 |
| BiPali (+LLM) | 56.5 ↓ -2.0 | 30.0 ↓ -2.9 | 67.4 ↓ -3.1 | 76.9 ↑ 14.2 | 33.4 ↑ 2.9 | 43.7 ↑ 17.2 | 71.2 ↓ -3.1 | 61.9 ↓ -11.7 | 73.8 ↓ -0.4 | 73.6 ↓ -8.8 | 58.8 ↑ 0.2 |
| *ColPali* (+Late Inter.) | **79.1** ↑ 22.6 | **54.4** ↑ 24.5 | **81.8** ↑ 14.4 | **83.9** ↑ 7.0 | **65.8** ↑ 32.4 | **73.2** ↑ 29.5 | **96.2** ↑ 25.0 | **91.0** ↑ 29.1 | **92.7** ↑ 18.9 | **94.4** ↑ 20.8 | **81.3** ↑ 22.5 |

Table 2: **Comprehensive evaluation of baseline models and our proposed method on *ViDoRe*.** Results are presented using NDCG@5 metrics, and illustrate the impact of different components. Text-only metrics are not computed for benchmarks with only visual elements.

## Can it read visual documents?

Same backbone and data: late interaction alone scores 38% higher on page images.

✗ Is a sum of maxima always right?




## Does it help coding agents?

Inside Codex: 14% more accurate, in half the tool calls.

✗ In domain, single vectors catch up


"Revisiting Text Ranking in Deep Research," 2026. Faysse et al., "ColPali," 2024. Mixedbread, 2026.

Part 4

# Bridging the gap to brute force

Which customers have a major unresolved problem with their order?

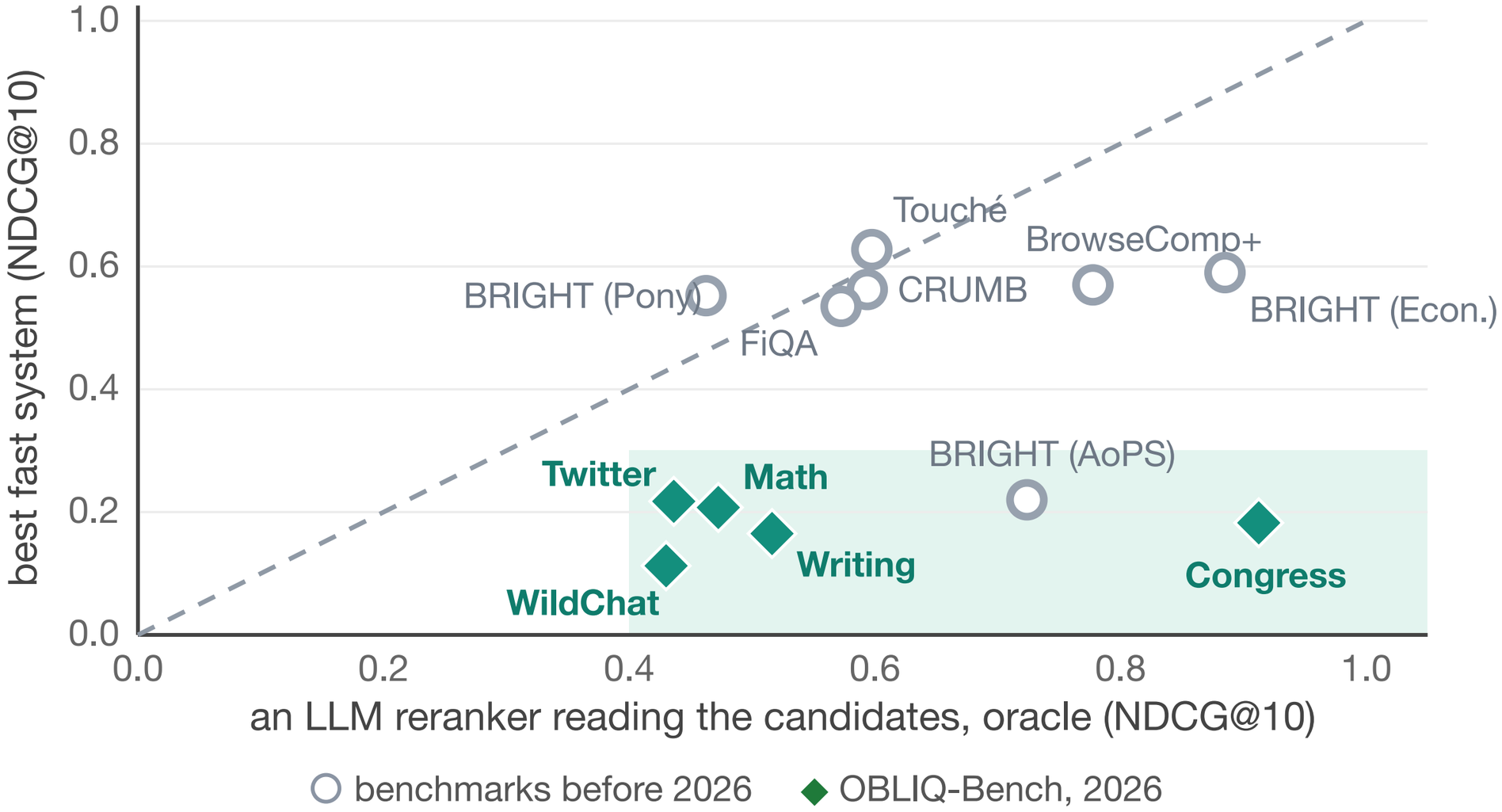
## What the index finds

| | |
|---|---|
| **Shopify** | A **problem** with your payout settings |
| **Kiln Supply** | Re: #5521 — the shelf arrived **broken** |

## What is actually there

| | |
|---|---|
| **Mark Chen** | **Re: #1032 — any news on the replacement? It's been a month** |
| **Sofia Reyes** | **Re: #1019 — following up again, it still hasn't come** |
| **Amir Haddad** | **Re: #998 — the whole set came in pieces. What now?** |

# Do we even need better search?



# What does an oblique query look like?

**Descriptive**

"Find all conversations where the AI's output shows **formatting corruption**, and the AI fails to self-correct."

**Analogue**

"Retrieve all problems that require the **same solution technique** as $y'' - 2y' + y = 0$, with $y(0) = 1,\ y'(0) = 1$."

**Tip of the tongue**

"I'm trying to find this Senate hearing clip: a woman starts out oddly friendly… Then she flips, and **pins him down** while he keeps saying 'I can't really speak to that.'"

They seek documents that instantiate a latent pattern, with no obvious surface expression.

# Externalize the context

```
$ grep -l "order #" inbox/* > orders
$ wc -l < orders
3512
$ grep -il "again\|still\|replacement" $(cat orders)
inbox/0927-mark-chen.txt
inbox/1002-sofia-reyes.txt
$ sed -n 1,12p inbox/0927-mark-chen.txt
```

The corpus stays in files.

The agent greps,
and combines clues.

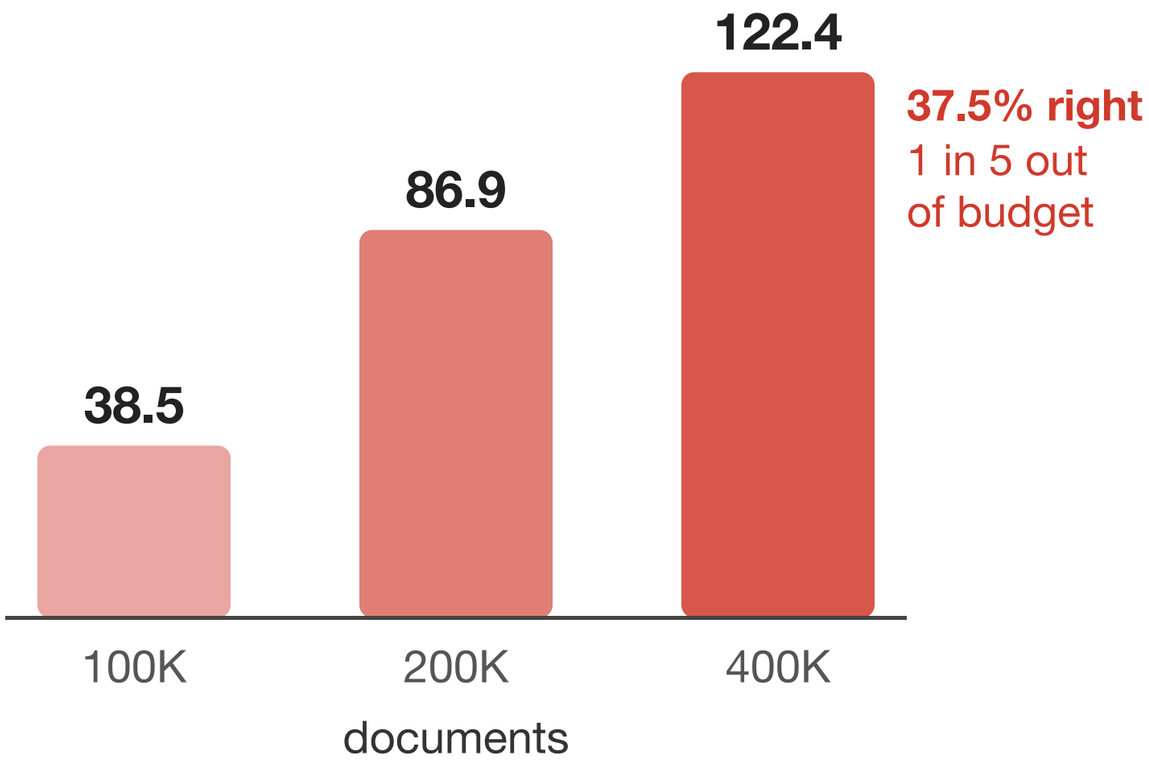
Only what it reads
enters its context.

BrowseComp-Plus, the same Claude Sonnet 4.6 agent: accuracy

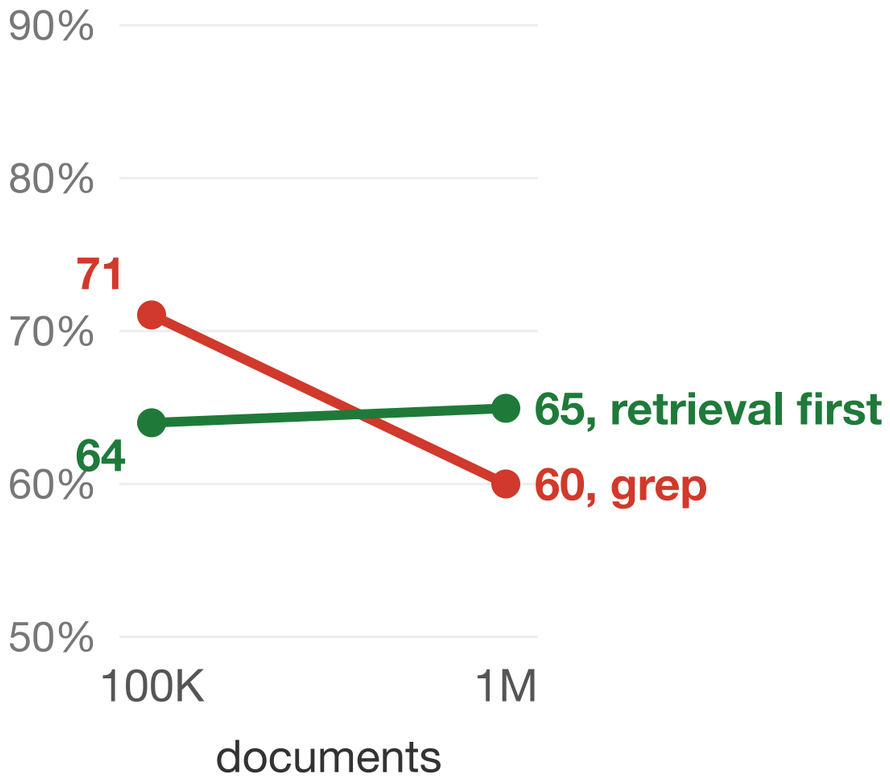


# What happens as the corpus grows?

**Grep over everything: searches per question**



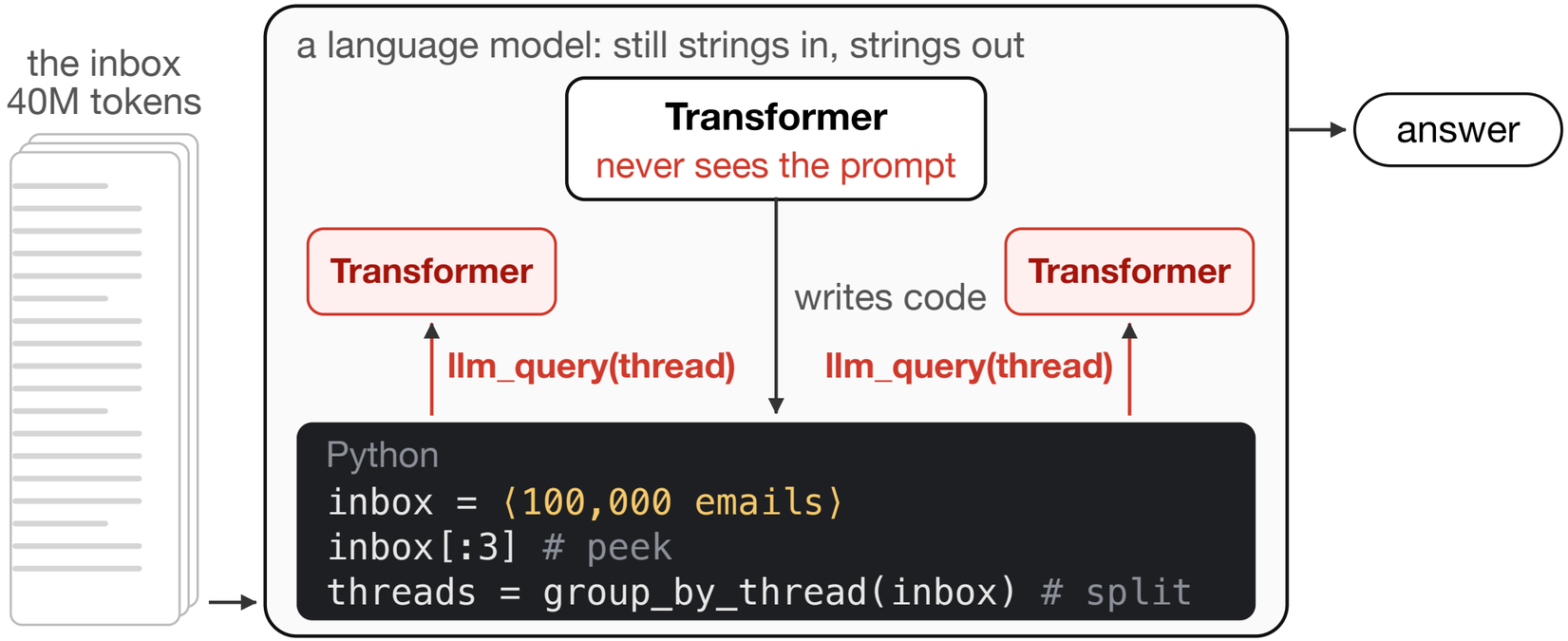

**Accuracy, the same small model**





Li, …, Lin, et al., 2026, RQ4. Zhuang, Ni, Fun, Lin, and Ma, 2026, Table 2.

# Recursive language models: externalize the context and allow recursion




1. Give the model a handle to the prompt, not the prompt.
2. Let it read and write in a REPL.
3. And call itself on new handles.

Which customers have a major unresolved problem with their order?

```
len(inbox), inbox[0]
→ 100000, Email(sender="ITA Airways", subject="Updated itinera…
```

**100,000** emails

```
threads = group_by_thread(inbox)
→ 41230
```

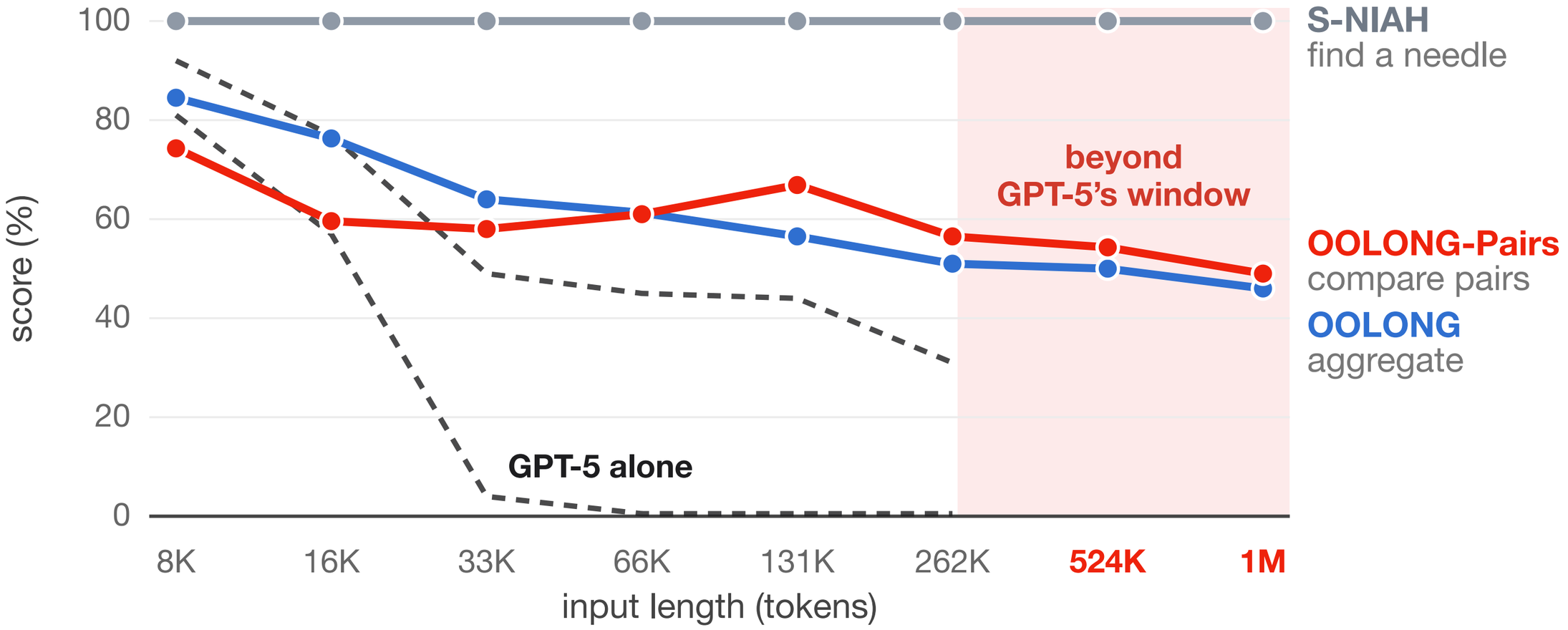
**41,230** threads

```
orders = [t for t in threads if "order #" in t.subject]
→ 3512
```

**3,512** about an order

```
open = [t for t in orders if llm_query(
    f"{t}\n\nIs there a major unresolved problem with the order?"
) == "yes"]
→ 23, after 3,512 sub-calls
```

**23** unresolved


100
80
60
40
20
0
score (%)
8K
16K
33K
66K
131K
262K
524K
1M
input length (tokens)
beyond
GPT-5's window
GPT-5 alone
S-NIAH
find a needle
OOLONG-Pairs
compare pairs
OOLONG
aggregate

Which of this week's orders look unusual?



→ **a usual order**

from every past order

Every past order counts, a little.

This needs studying the corpus, not just searching it.

**Lecture 9: learning in weights**

# Today

✓ **Massive contexts in practice**

Split the signature: index once, then answer every question.

✓ **Neural information retrieval**

Cross-encoders read with the question. Dual encoders encode ahead of time.

✓ **Beyond exhaustive search**

About $\sqrt{N}$ or $\log N$ dot products. A vector per token, about 20 bytes each.

✓ **Bridging the gap to brute force**

Reading with the question still wins. Retrieval can bound it, and RLMs recurse over it.

NEXT WEEK

# How do we know whether an AI system **works**?

Which inputs? Who decides what is right?
What can a score tell us, and what can't it?

**With Quiz 3.**