Data-Intensive Distributed Computing
CS 451/651 (Fall 2025)

UNIVERSITY OF WATERLOO

Text Processing II
(v1.01)

Week 10: November 4, 2025

Jimmy Lin
David R. Cheriton School of Computer Science
University of Waterloo

These slides are available at http://lintool.github.io/bigdata-2025f/

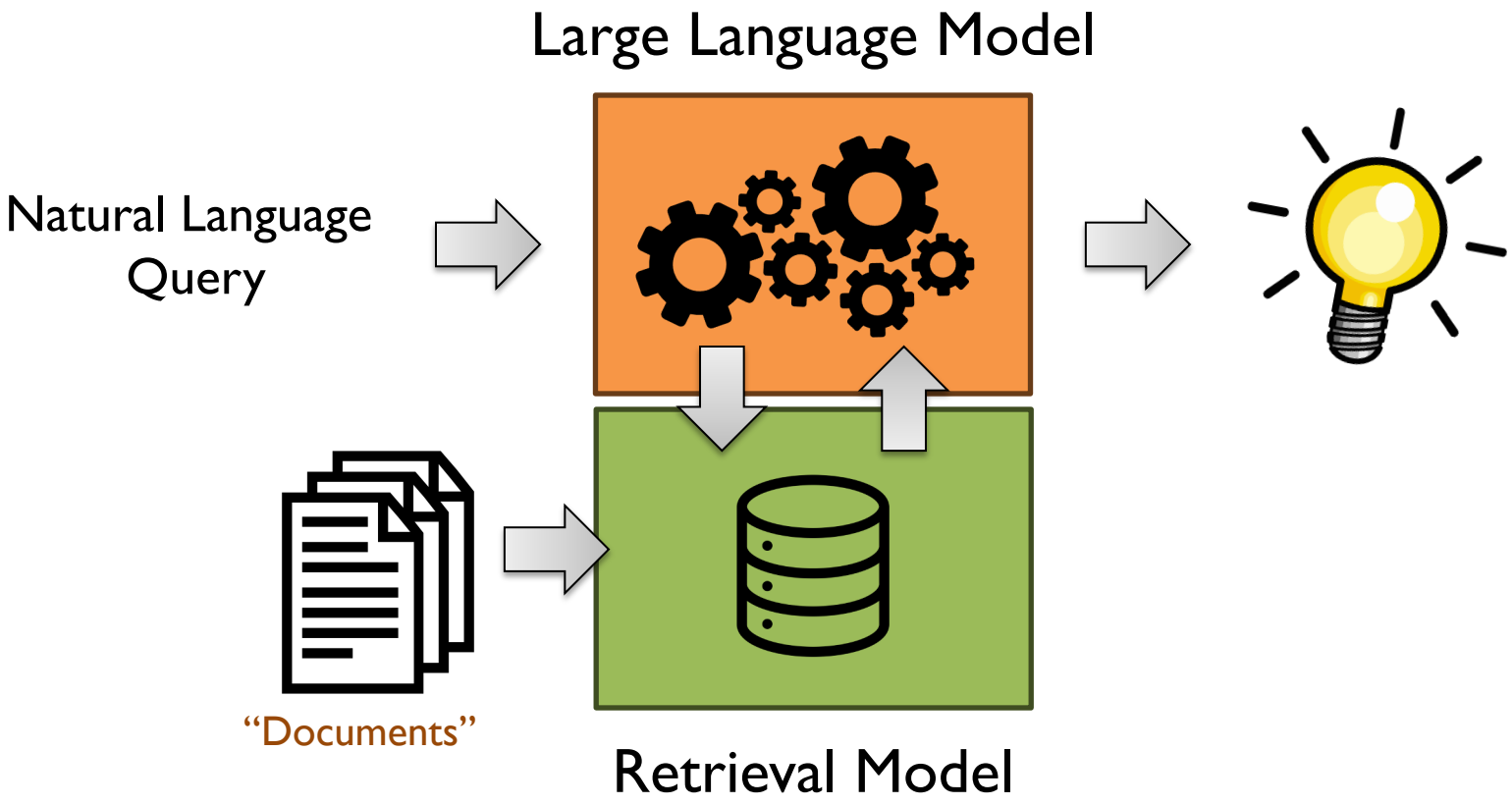
Key Questions

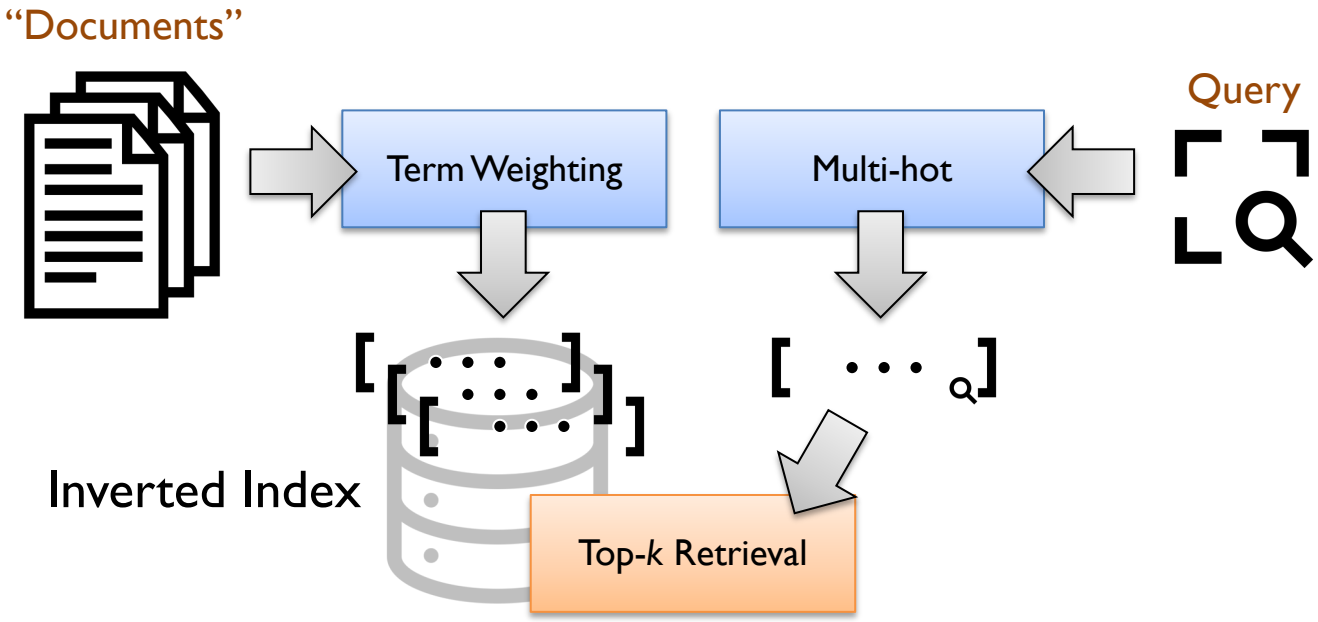
For sparse vector representations, how do we assign weights?
For sparse retrieval, how do we perform top-*k* retrieval efficiently?

For dense vector representations, how do we assign weights?
For dense retrieval, how do we perform top-*k* retrieval efficiently?

Do we need specialized databases for dense and sparse vectors?

Retrieval-Augmented Generation

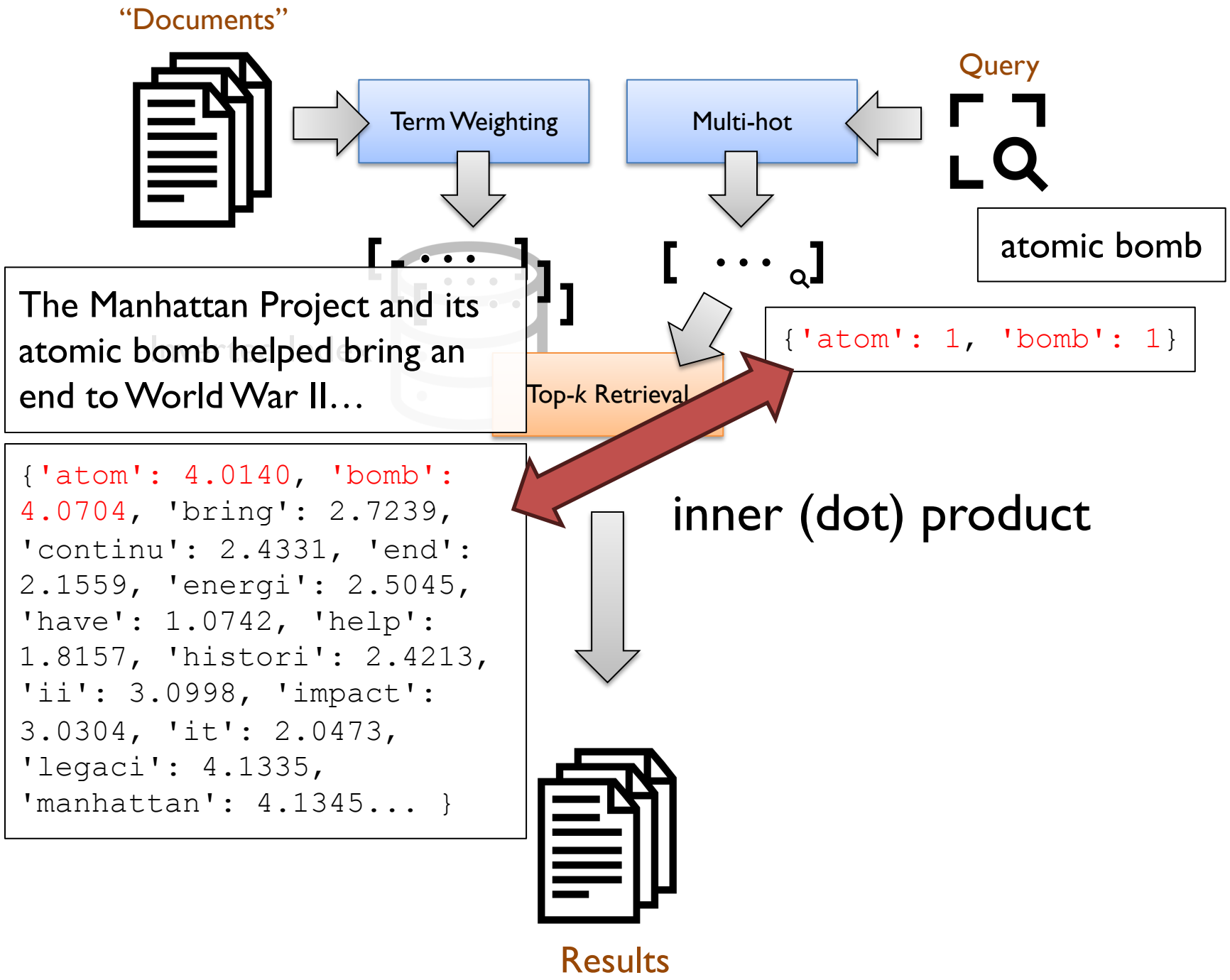




Challenge #1: How do we assign weights?

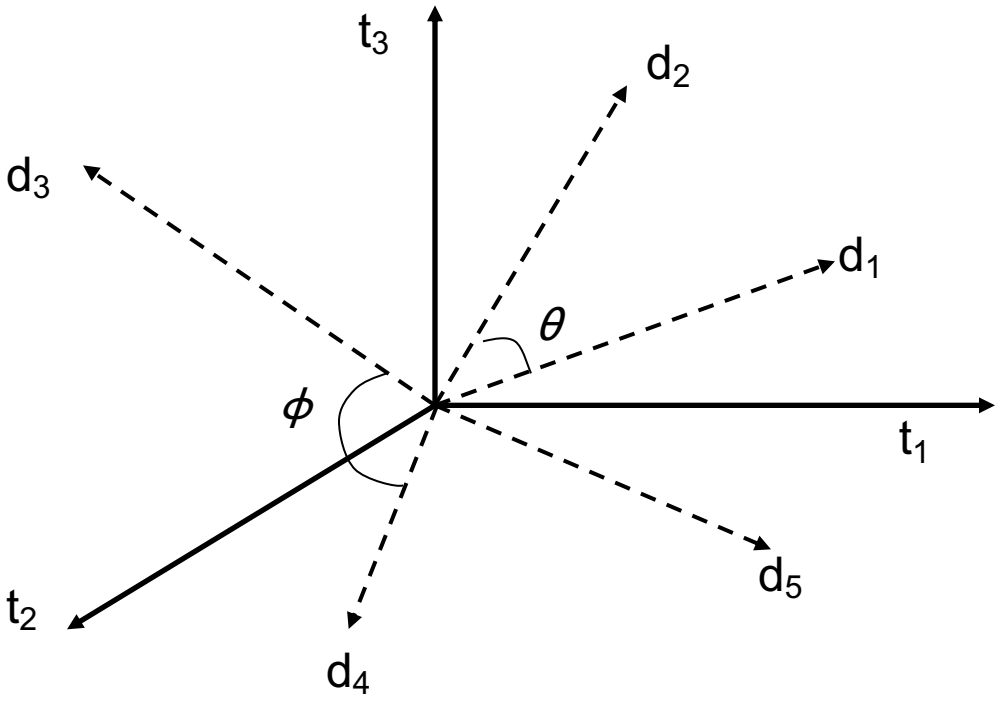
Challenge #2: How do we perform top-*k* retrieval efficiently?

Results


"Documents"
Term Weighting
Multi-hot
Query
atomic bomb
The Manhattan Project and its atomic bomb helped bring an end to World War II…
{'atom': 1, 'bomb': 1}
Top-k Retrieval
{'atom': 4.0140, 'bomb': 4.0704, 'bring': 2.7239, 'continu': 2.4331, 'end': 2.1559, 'energi': 2.5045, 'have': 1.0742, 'help': 1.8157, 'histori': 2.4213, 'ii': 3.0998, 'impact': 3.0304, 'it': 2.0473, 'legaci': 4.1335, 'manhattan': 4.1345... }
inner (dot) product
Results

Why?

Vector Space Model



Assumption: Documents that are "close together" in vector space "talk about" the same things

Therefore, retrieve documents based on how close the document is to the query (i.e., similarity ~ "closeness")

Why?

tl;dr – inner products reflect vector similarity

Similarity Metric

Use "angle" between the vectors:

$$d_j = [w_{j,1}, w_{j,2}, w_{j,3}, \ldots w_{j,n}]$$
$$d_k = [w_{k,1}, w_{k,2}, w_{k,3}, \ldots w_{k,n}]$$

$$\cos\theta = \frac{d_j \cdot d_k}{|d_j||d_k|}$$

$$\text{sim}(d_j, d_k) = \frac{d_j \cdot d_k}{|d_j||d_k|} = \frac{\sum_{i=0}^{n} w_{j,i} w_{k,i}}{\sqrt{\sum_{i=0}^{n} w_{j,i}^2} \sqrt{\sum_{i=0}^{n} w_{k,i}^2}}$$

Or, more generally, inner products:

$$\text{sim}(d_j, d_k) = d_j \cdot d_k = \sum_{i=0}^{n} w_{j,i} w_{k,i}$$

Challenge #1: Weight Assignment

Intuition:

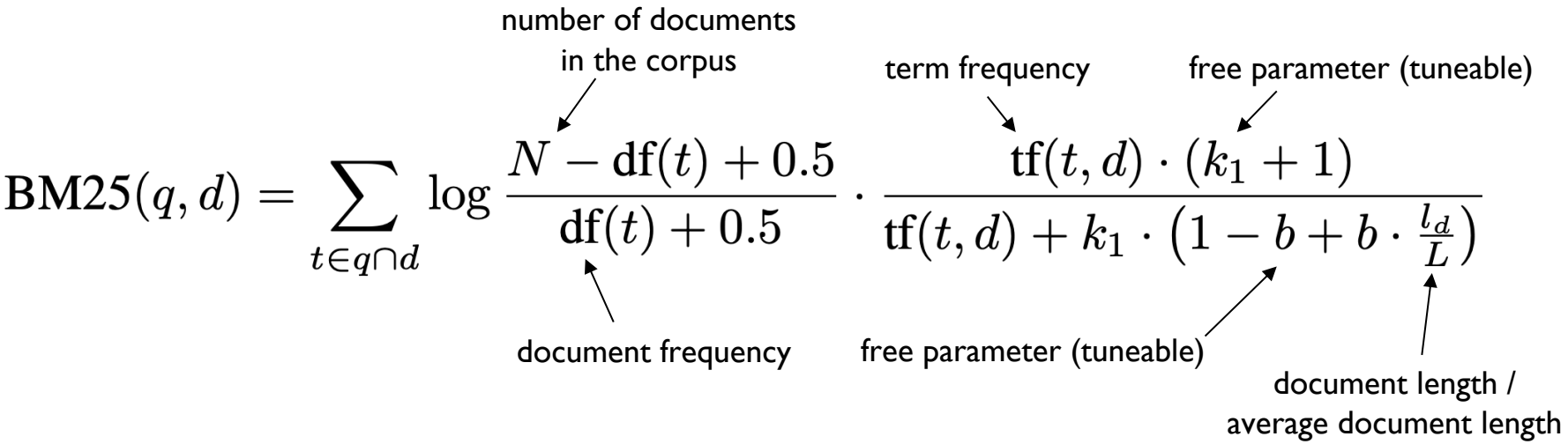
Local: terms that appear often in a document → high weights

Global: terms that appear in many documents → low weights

How do we capture this mathematically?

Term frequency (local)

Document frequency (global)



Challenge #2: Efficient Top-*k* Retrieval

The Manhattan Project and its atomic bomb helped bring an end to World War II…

atomic bomb

```
{'atom': 4.0140, 'bomb': 4.0704, 'bring': 2.7239, 'continu': 2.4331, 'end': 2.1559, 'energi': 2.5045, 'have': 1.0742, 'help': 1.8157, 'histori': 2.4213, 'ii': 3.0998, 'impact': 3.0304, 'it': 2.0473, 'legaci': 4.1335, 'manhattan': 4.1345... }
```

```
{'atom': 1, 'bomb': 1}
```

$$\text{sim}(d_j, d_k) = d_j \cdot d_k = \sum_{i=0}^{n} w_{j,i} w_{k,i}$$

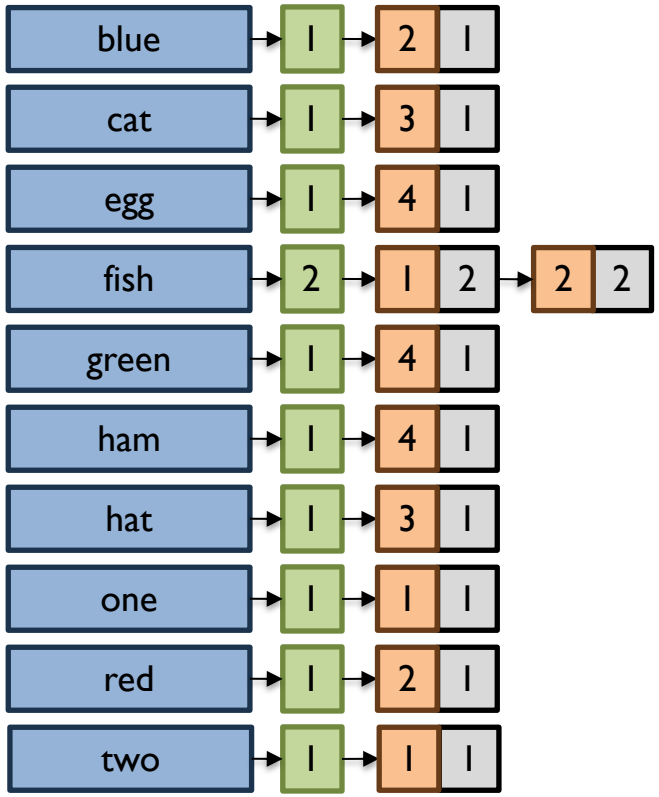
Don't actually store the weights in the inverted index!

Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham



$$\text{BM25}(q,d) = \sum_{t \in q \cap d} \log \frac{N - \text{df}(t) + 0.5}{\text{df}(t) + 0.5} \cdot \frac{\text{tf}(t,d) \cdot (k_1 + 1)}{\text{tf}(t,d) + k_1 \cdot \left(1 - b + b \cdot \frac{l_d}{L}\right)}$$

Assume everything fits in memory on a single machine…

(For now)

Retrieval in a Nutshell

Look up postings lists corresponding to query terms

Traverse postings for each query term

Store partial query-document scores in accumulators

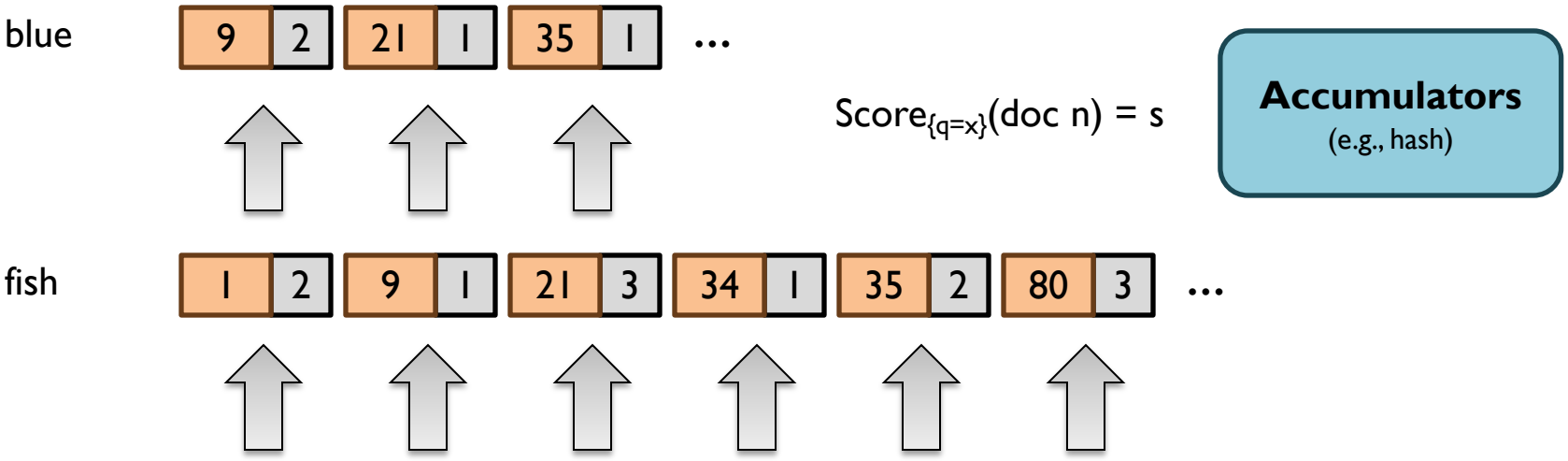
Select top-*k* results to return

$$\text{BM25}(q,d) = \sum_{t \in q \cap d} \log \frac{N - \text{df}(t) + 0.5}{\text{df}(t) + 0.5} \cdot \frac{\text{tf}(t,d) \cdot (k_1 + 1)}{\text{tf}(t,d) + k_1 \cdot \left(1 - b + b \cdot \frac{l_d}{L}\right)}$$

Retrieval: Term-at-a-Time

Evaluate documents one query term at a time

Usually, starting from most rare term



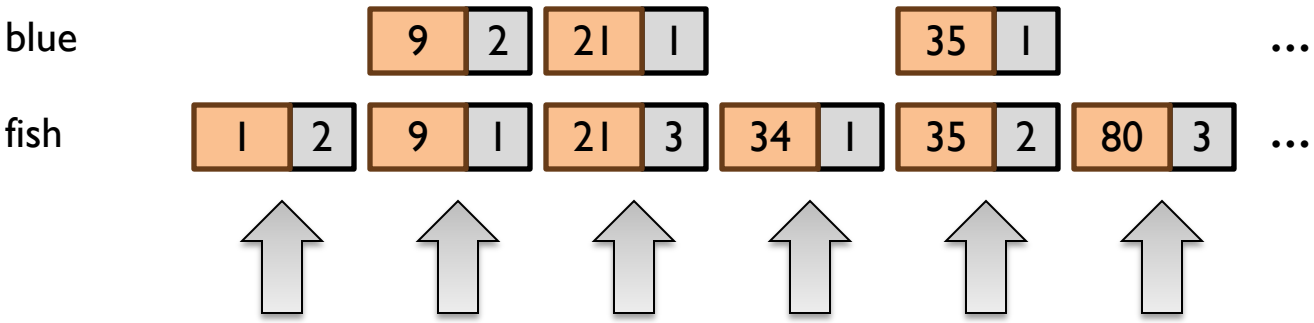
Tradeoffs:

Early termination heuristics (good)

Large memory footprint (bad), but filtering heuristics possible

Retrieval: Document-at-a-Time

Evaluate documents one at a time (score all query terms)



Accumulators
(e.g. min heap)

Document score in top *k*?

Yes: Insert document score, extract-min if heap too large
No: Do nothing

Tradeoffs:

Small memory footprint (good)
Skipping possible to avoid reading all postings (good)
More seeks and irregular data accesses (bad)

Assume everything fits in memory on a single machine…

Now let's relax this constraint

Important Ideas

Partitioning (for scalability)

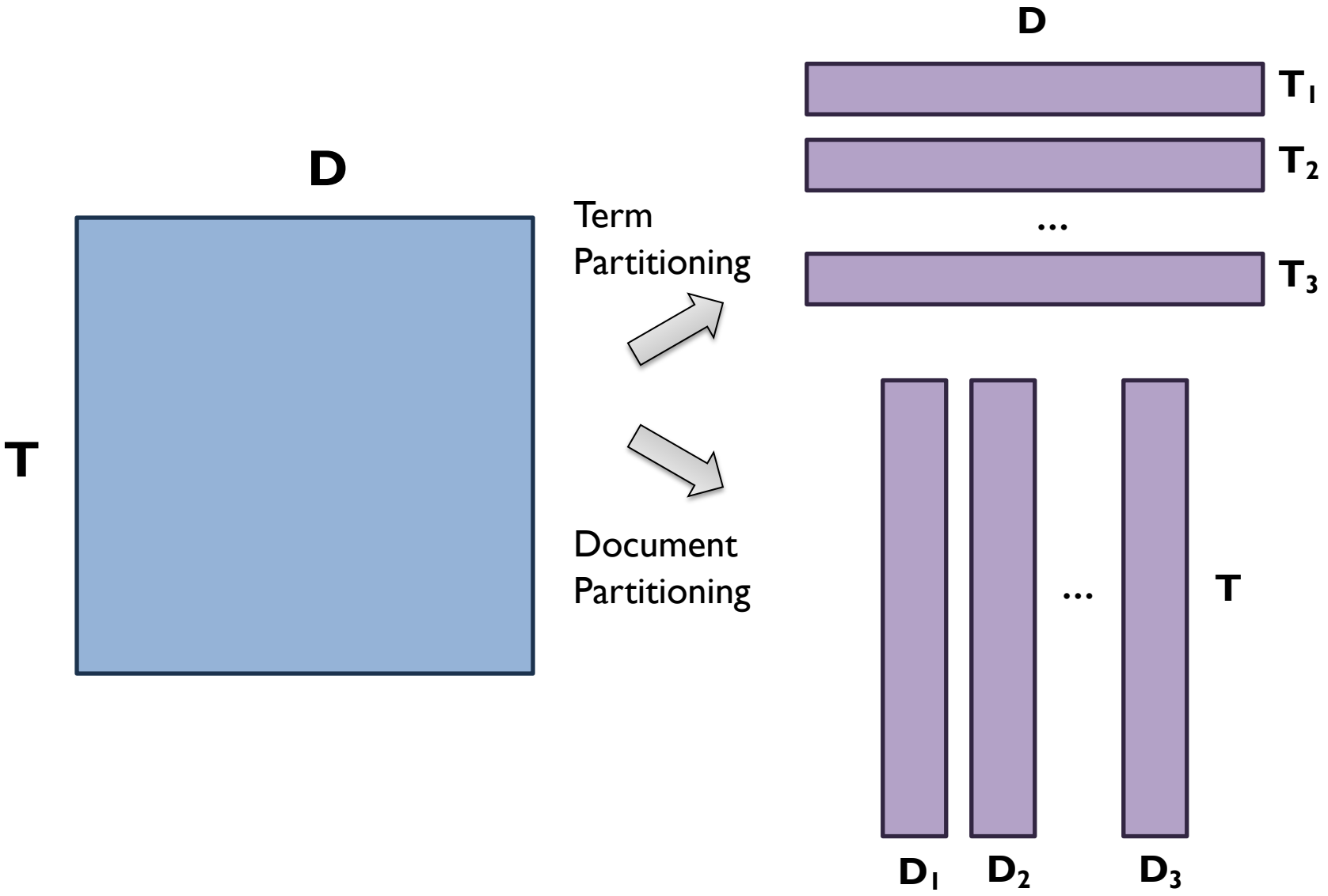
Replication (for redundancy)

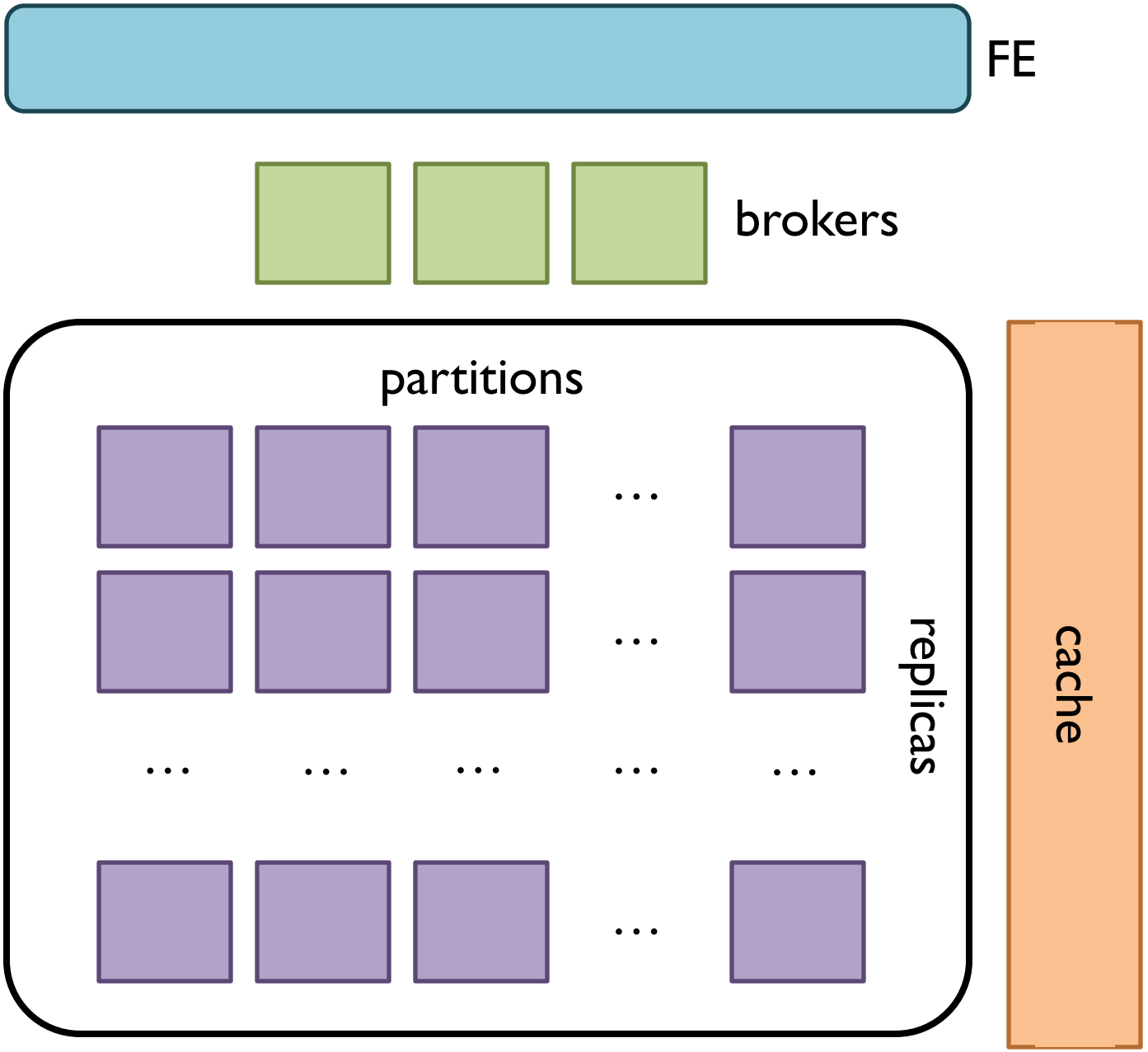
Caching (for speed)

Routing (for load balancing)

The rest is just details!

Term vs. Document Partitioning




FE
brokers
partitions
replicas
cache

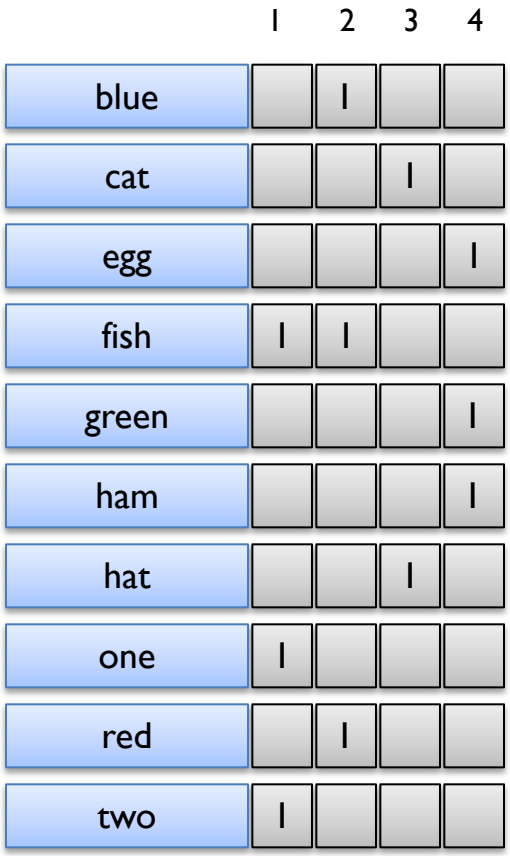
Datacenter
brokers
partitions
Tier
replicas
cache
Datacenter
brokers
partitions
Tier
replicas
cache
Datac
part
Tier

Doc 1
one fish, two fish

Doc 2
red fish, blue fish

Doc 3
cat in the hat

Doc 4
green eggs and ham



Indexing: building this structure

Retrieval: using it to perform top-*k* retrieval

How?

~~(1) Let's learn how to actually do it…~~

(2) Call an external package

RAG / Lakehouse Integration: Why?

Ingesting multiple sources of data

User-generated content + behavioral data

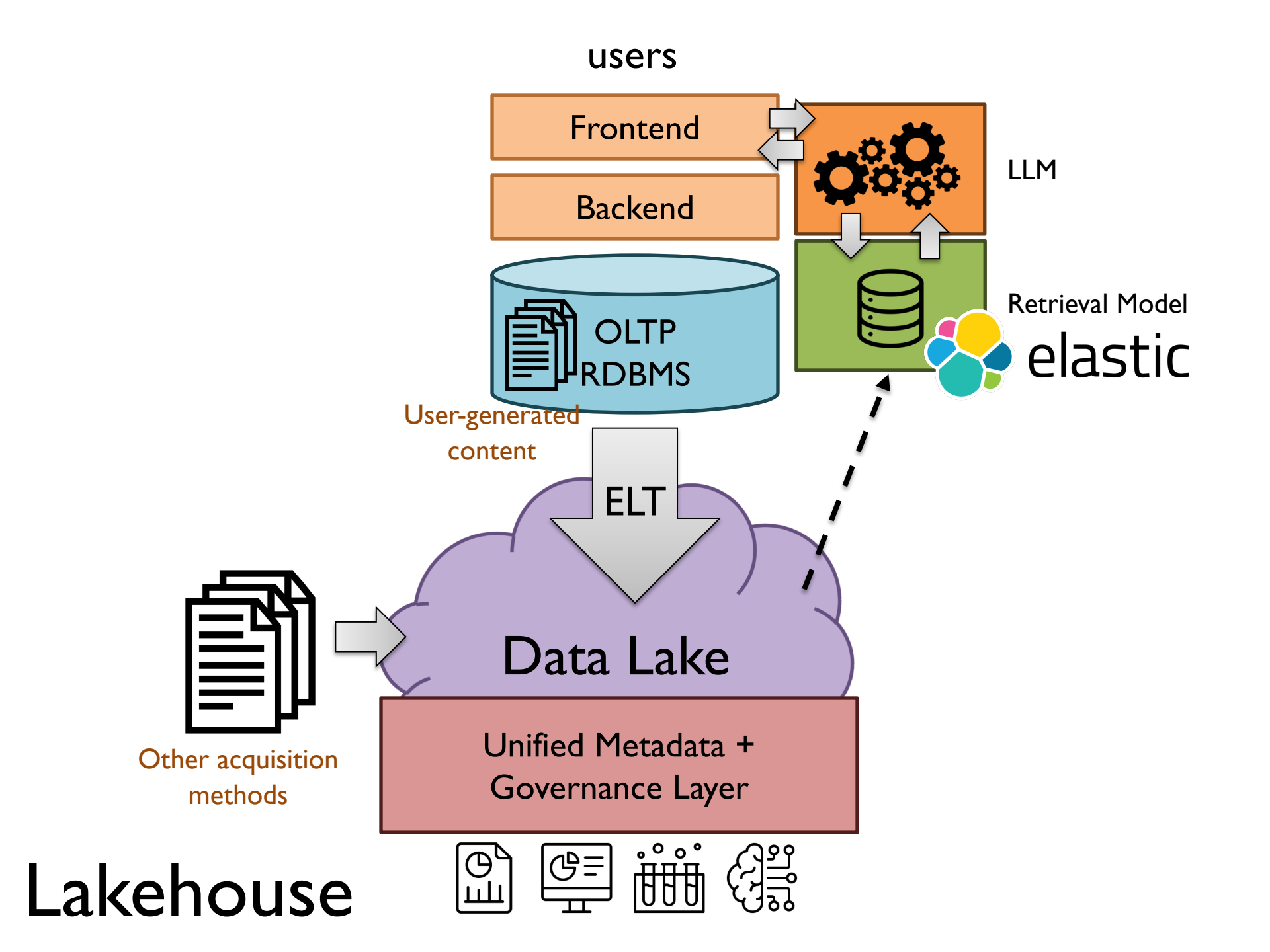
Other sources…

Performing "standard" lakehouse tasks

Joining, filtering, projecting, etc. heterogenous data

Data cleaning, aggregation, etc.

Training ML models


users
Frontend
Backend
LLM
OLTP
RDBMS
Retrieval Model
elastic
User-generated
content
ELT
Data Lake
Unified Metadata +
Governance Layer
Other acquisition
methods
Lakehouse

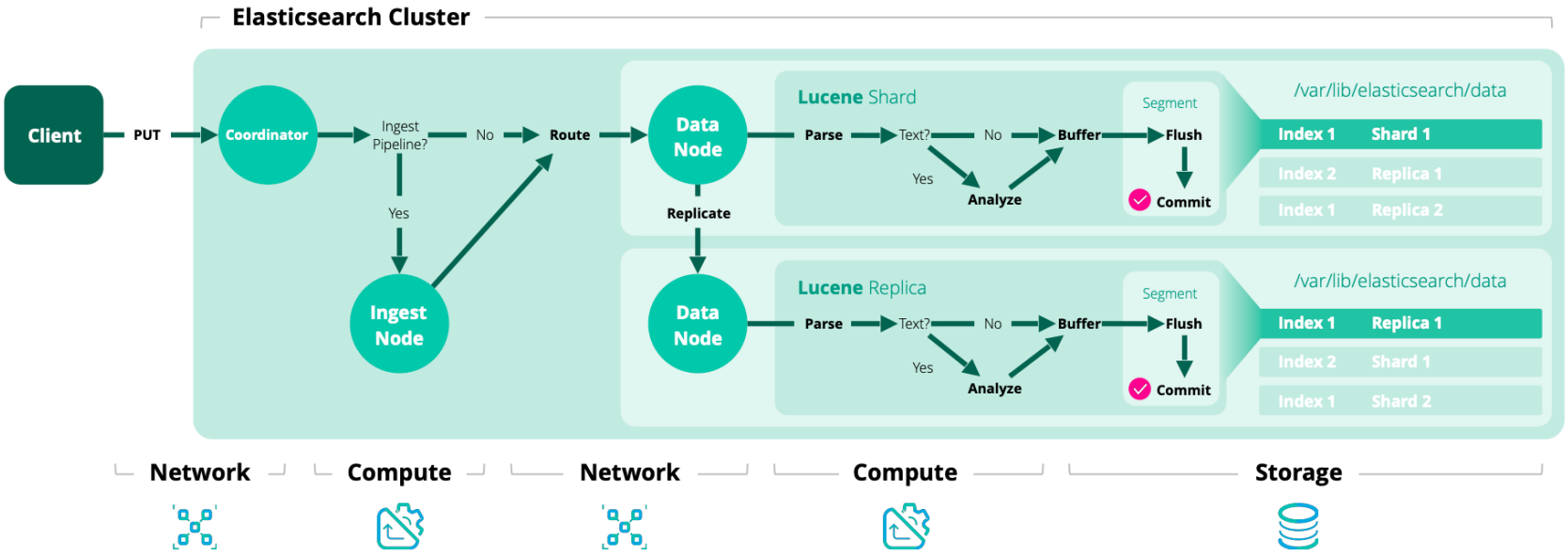
POST /{index}/_search Console

```
GET /my-index-000001/_search?from=40&size=20
{
  "query": {
    "term": {
      "user.id": "kimchy"
    }
  }
}
```

POST /{index}/_create/{id} Console

```
PUT my-index-000001/_create/1
{
  "@timestamp": "2099-11-15T13:12:00",
  "message": "GET /search HTTP/1.1 200 1070000",
  "user": {
    "id": "kimchy"
  }
}
```

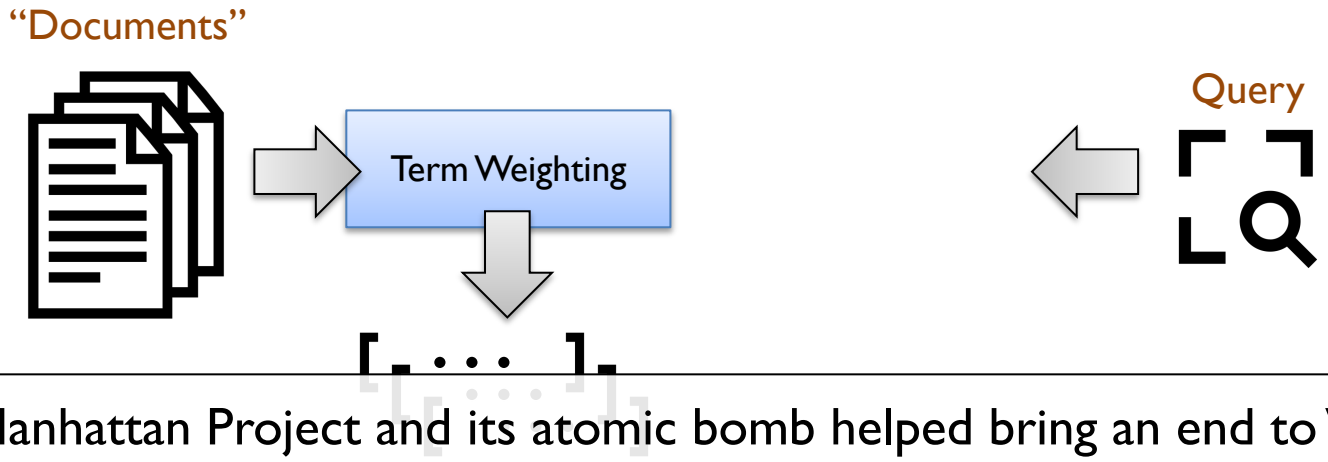
https://www.elastic.co/docs/api/doc/elasticsearch/operation/operation-search
https://www.elastic.co/docs/api/doc/elasticsearch/operation/operation-create



Every indexing operation in Elasticsearch is first resolved to a replication group using routing, typically based on the document ID. Once the replication group has been determined, the operation is forwarded internally to the current primary shard of the group. This stage of indexing is referred to as the coordinating stage.

The next stage of indexing is the primary stage, performed on the primary shard. The primary shard is responsible for validating the operation and forwarding it to the other replicas…

https://www.elastic.co/docs/deploy-manage/distributed-architecture/reading-and-writing-documents

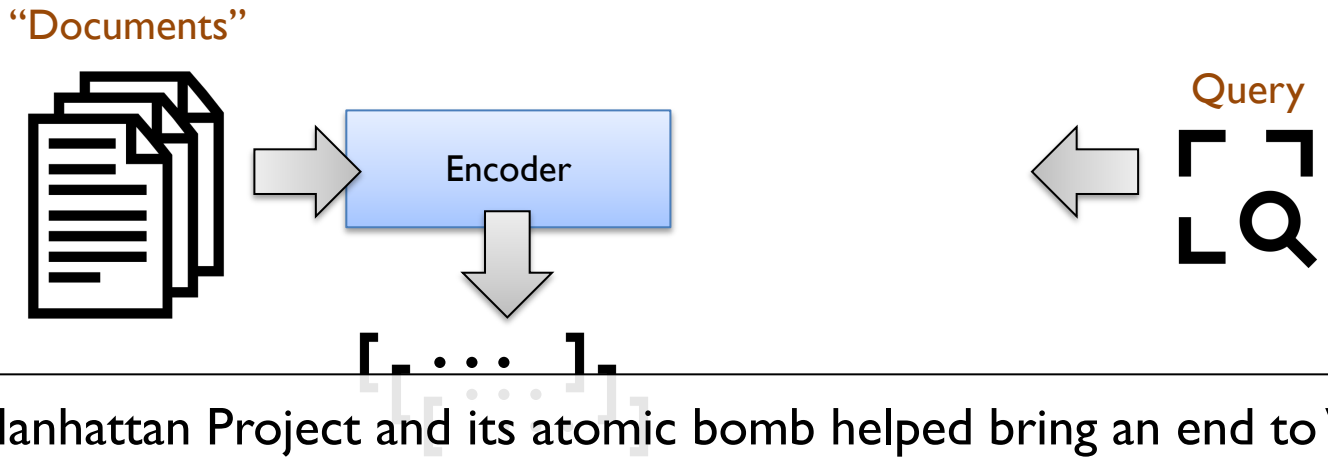


The Manhattan Project and its atomic bomb helped bring an end to World War II. Its legacy of peaceful uses of atomic energy continues to have an impact on history and science.

```
{'atom': 4.0140, 'bomb': 4.0704, 'bring': 2.7239, 'continu':
2.4331, 'end': 2.1559, 'energi': 2.5045, 'have': 1.0742, 'help':
1.8157, 'histori': 2.4213, 'ii': 3.0998, 'impact': 3.0304, 'it':
2.0473, 'legaci': 4.1335, 'manhattan': 4.1345, 'peac': 3.5205,
'project': 2.6442, 'scienc': 2.8700, 'us': 0.9967, 'war':
2.6454, 'world': 1.9974}
```

sparse vector

Results



The Manhattan Project and its atomic bomb helped bring an end to World War II. Its legacy of peaceful uses of atomic energy continues to have an impact on history and science.

```
[0.099843978881836, 0.870057582855224 6, 0.520509719848633,
0.030491352081299, 0.723929882049560 5, 0.134523391723633,
0.433127403259277 3, 0.644286632537842, 0.645430564880371,
0.047342777252197 3, 0.070496082305908, 0.504533529281616,
0.815732955932617 2, 0.133575916290283, 0.997444868087768 6,
0.074254274368286 1, 0.155941247940063 5, 0.421395778656006,
0.014032363891602, 0.996794581413269...]
```

dense vector

Results

Why?

Semantic Retrieval: Challenges

The Manhattan Project and its atomic bomb helped bring an end to World War II. Its legacy of peaceful uses of atomic energy continues to have an impact on history and science.

```
[0.099843978881836, 0.8700575828552246, 0.520509719848633,
0.030491352081299, 0.7239298820495605, 0.134523391723633,
0.4331274032592773, 0.644286632537842, 0.645430564880371,
0.047342772521973, 0.070496082305908, 0.504533529281616,
0.815732955932617 2, 0.133575916290283, 0.997444868087768 6,
0.074254274368286 1, 0.155941247940063 5, 0.421395778656006,
0.014032363891602, 0.996794581413269...]
```

Challenge #1: How do we assign weights?

Challenge #2: How do we perform top-*k* retrieval efficiently?

Challenge #1: Weight Assignment

Q: Where do these dense vectors (= embeddings) come from?

A: They're learned from data!

Let $\mathcal{D} = \{\langle q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^- \rangle\}_{i=1}^m$ be the training data that consists of m instances. Each instance contains one question q_i and one relevant (positive) passage p_i^+, along with n irrelevant (negative) passages $p_{i,j}^-$. We optimize the loss function as the negative log likelihood of the positive passage:

$$L(q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^-) \quad (2)$$

$$= -\log \frac{e^{\text{sim}(q_i, p_i^+)}}{e^{\text{sim}(q_i, p_i^+)} + \sum_{j=1}^n e^{\text{sim}(q_i, p_{i,j}^-)}}.$$

Intuition: you have many queries with relevant and non-relevant examples

inner product of (queries, relevant examples) → make this high

inner product of (queries, non-relevant examples) → make this low

Karpukhin et al. Dense Passage Retrieval for Open-Domain Question Answering. *EMNLP 2020*.

Remember this?

The Task

Given: $D = \{(\mathrm{x}_i, y_i)\}_i^n$

label

feature vector

$$\mathrm{x}_i = [x_1, x_2, x_3, \ldots, x_d]$$
$$y \in \{0, 1\}$$

Induce: $f : X \rightarrow Y$

Such that loss is minimized

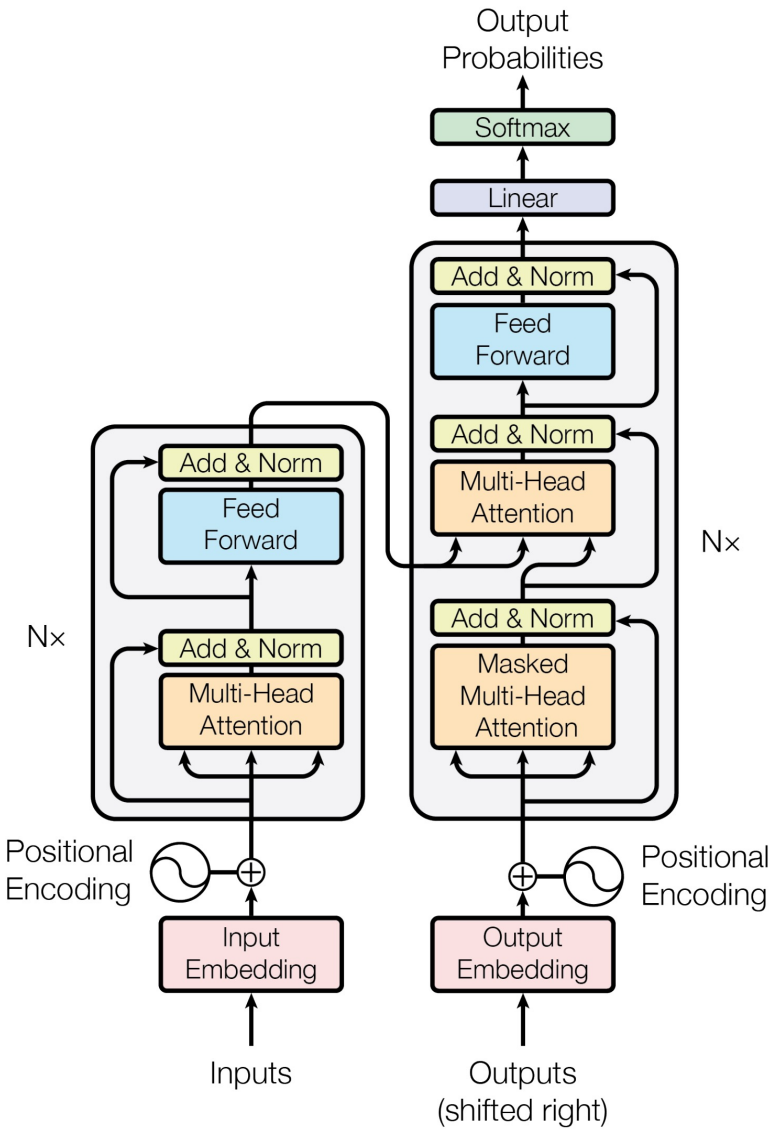
$$\frac{1}{n} \sum_{i=0}^{n} \ell(f(\mathrm{x}_i), y_i)$$

loss function

Typically, we consider functions of a parametric form:

$$\arg\min_{\theta} \frac{1}{n} \sum_{i=0}^{n} \ell(f(x_i; \theta), y_i)$$

model parameters



Transformers!

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

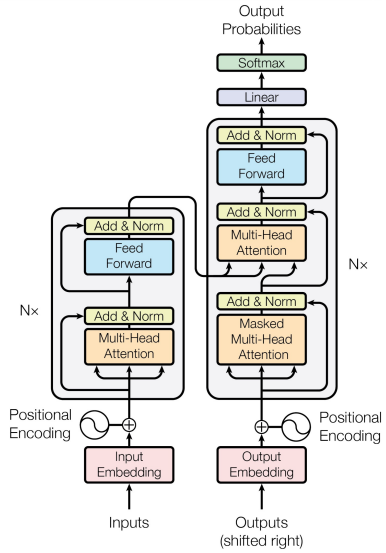
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.com

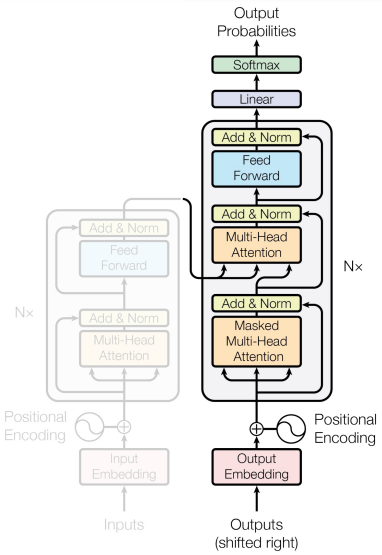
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task,

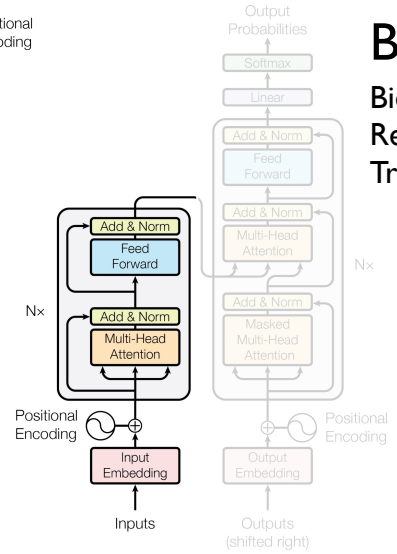


Transformer (2017)



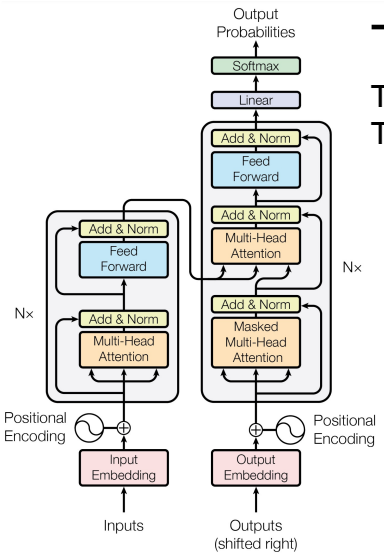
GPT (2018)

Generative Pretrained Transformer



BERT (2018)

Bidirectional Encoder Representations from Transformers

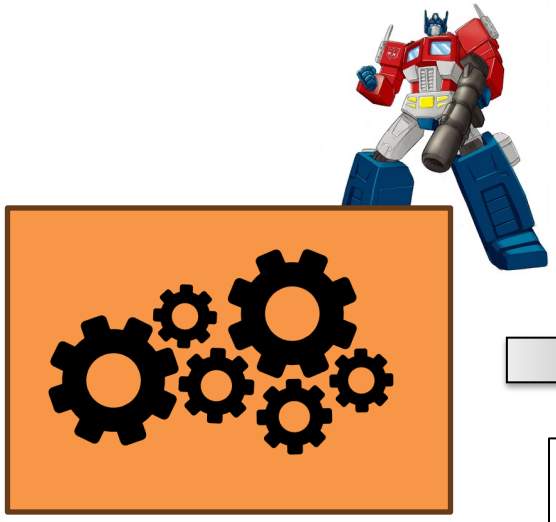


T5 (2019)

Text-To-Text Transfer Transformer

The Manhattan Project and its atomic bomb helped bring an end to World War II. Its legacy of peaceful uses of atomic energy continues to have an impact on history and science.

Input



Model

Output

```
[0.099843978881836, 0.8700575828552246,
0.520509719848633, 0.030491352081299,
0.7239298820495605, 0.134523391723633,
0.4331274032592773, 0.644286632537842,
0.645430564880371, 0.0473427772521973,
0.070496082305908, 0.504533529281616,
0.8157329559326172, 0.133575916290283,
0.9974448680877686, 0.0742542743682861,
0.1559412479400635, 0.421395778656006,
0.014032363891602, 0.996794581413269...]
```

Challenge #1: Weight Assignment

Q: Where do these dense vectors (= embeddings) come from?

A: They're learned from data

Let $\mathcal{D} = \{\langle q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^- \rangle\}_{i=1}^m$ be the training data that consists of m instances. Each instance contains one question q_i and one relevant (positive) passage p_i^+, along with n irrelevant (negative) passages $p_{i,j}^-$. We optimize the loss function as the negative log likelihood of the positive passage:

$$L(q_i, p_i^+, p_{i,1}^-, \cdots, p_{i,n}^-) \quad (2)$$
$$= -\log \frac{e^{\text{sim}(q_i, p_i^+)}}{e^{\text{sim}(q_i, p_i^+)} + \sum_{j=1}^n e^{\text{sim}(q_i, p_{i,j}^-)}}.$$

How do we actually do it?

Intuition: you get many queries with relevant and non-relevant examples

inner product of (queries, relevant examples) → make this high

inner product of (queries, non-relevant examples) → make this low

Karpukhin et al. Dense Passage Retrieval for Open-Domain Question Answering. *EMNLP 2020*.

How do we do it?

```
model.fit(q, p_pos, p_neg)
```

Challenge #2: Efficient Top-*k* Retrieval

The Manhattan Project and its atomic bomb helped bring an end to World War II. Its legacy of peaceful uses of atomic energy continues to have an impact on history and science.

```
{'atom': 4.0140, 'bomb': 4.0704, 'bring': 2.7239, 'continu':
2.4331, 'end': 2.1559, 'energi': 2.5045, 'have': 1.0742, 'help':
1.8157, 'histori': 2.4213, 'ii': 3.0998, 'impact': 3.0304, 'it':
2.0473, 'legaci': 4.1335, 'manhattan': 4.1345, 'peac': 3.5205,
'project': 2.6442, 'scienc': 2.8700, 'us': 0.9967, 'war':
2.6454, 'world': 1.9974}
```

Inverted indexes don't work…

```
[0.099843978881836, 0.8700575828552246, 0.52050971984863 3,
0.030491352081299, 0.7239298820495605, 0.134523391723633,
0.4331274032592773, 0.644286632537842, 0.645430564880371,
0.047342777252197 3, 0.070496082305908, 0.504533529281616,
0.815732955932617 2, 0.133575916290283, 0.997444868087768 6,
0.074254274368286 1, 0.155941247940063 5, 0.421395778656006,
0.014032363891602, 0.996794581413269...]
```

Challenge #2: Efficient Top-*k* Retrieval

Why don't inverted indexes work for dense vectors?

Inverted indexes are compact…

Because they only need to sort term frequencies

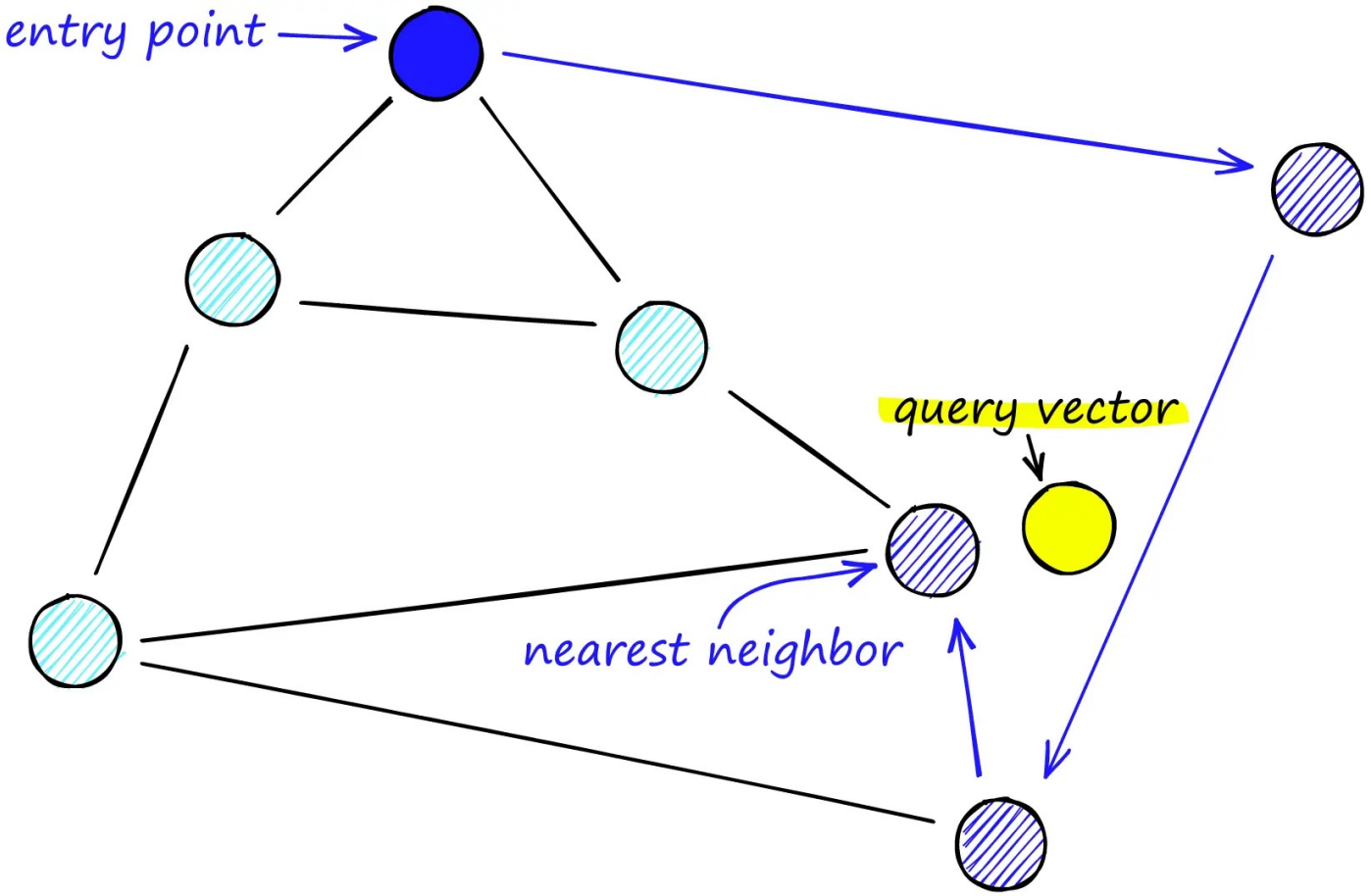
Because terms are sparse

Retrieval with inverted indexes is efficient…

Because queries are sparse

Challenge #2: Efficient Top-*k* Retrieval

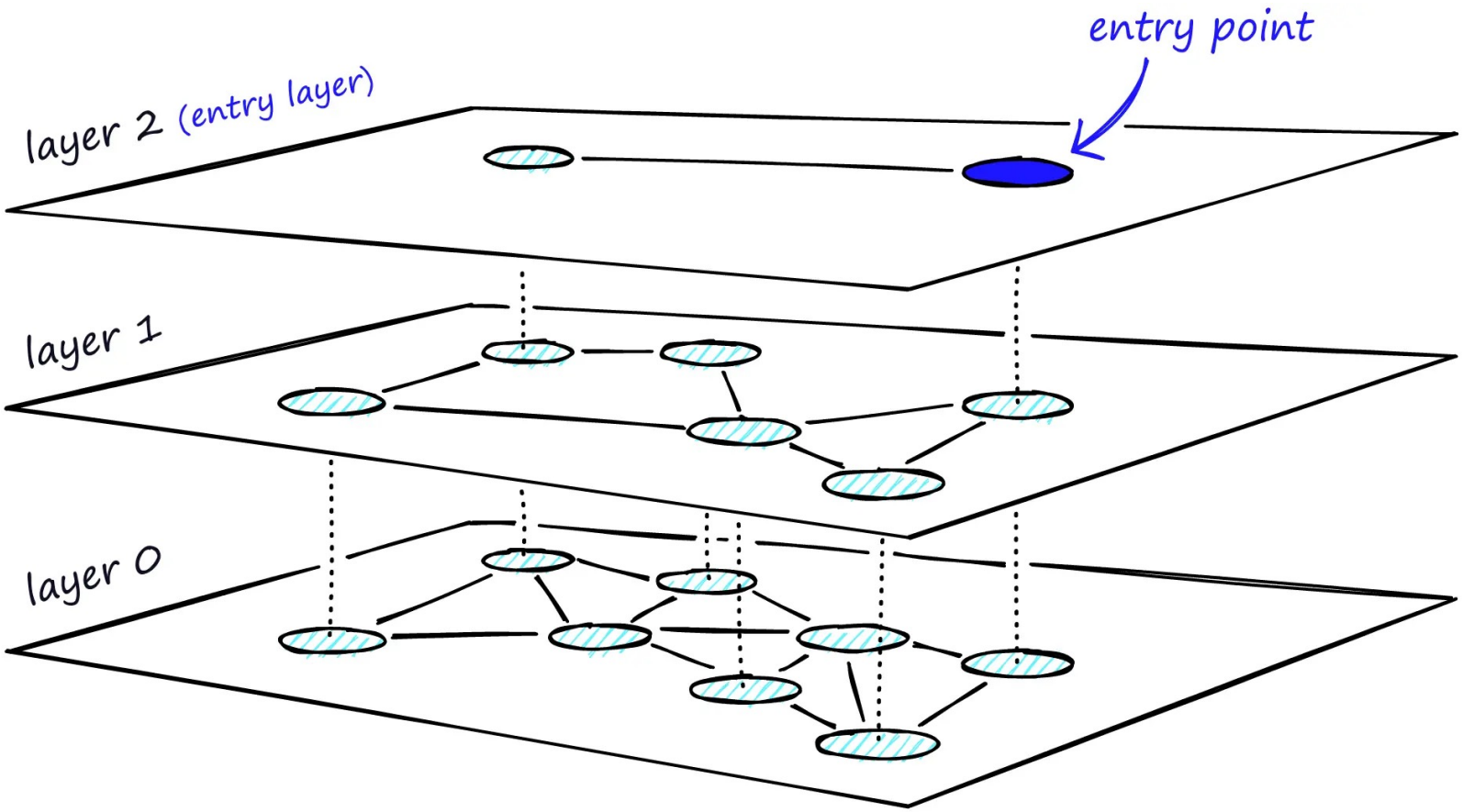
Use Hierarchical Navigable Small Worlds (HNSW) Indexes



https://www.elastic.co/docs/deploy-manage/distributed-architecture/reading-and-writing-documents

Challenge #2: Efficient Top-*k* Retrieval

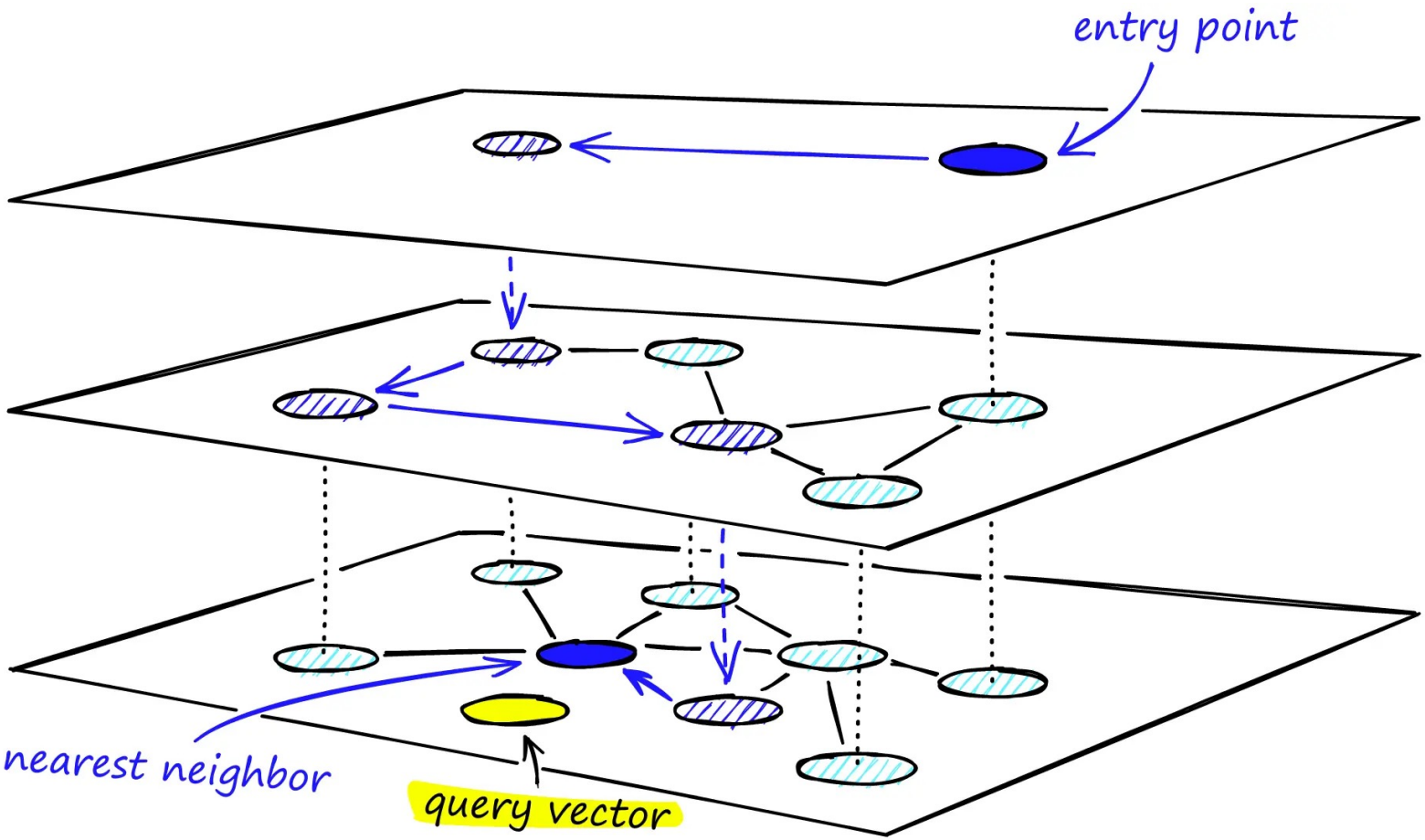
Use Hierarchical Navigable Small Worlds (HNSW) Indexes



https://www.elastic.co/docs/deploy-manage/distributed-architecture/reading-and-writing-documents

Challenge #2: Efficient Top-*k* Retrieval

Use Hierarchical Navigable Small Worlds (HNSW) Indexes



https://www.elastic.co/docs/deploy-manage/distributed-architecture/reading-and-writing-documents

Challenge #2: Efficient Top-*k* Retrieval

Use Hierarchical Navigable Small Worlds (HNSW) Indexes

Important characteristics:

Approximate

Non-deterministic between index builds

Alternatives?

Challenge #2: Efficient Top-*k* Retrieval

Use "flat indexes"

Simplest thing that can possibly work!

Store vectors sequentially

Retrieval = brute force scan across all vectors, computing similarities

Not a bad solution for "small" corpora (e.g., <1M docs)

Lin. Operational Advice for Dense and Sparse Retrievers: HNSW, Flat, or Inverted Indexes? *ACL 2025*.

RAG / Lakehouse Integration: Why?

Ingesting multiple sources of data

User-generated content + behavioral data

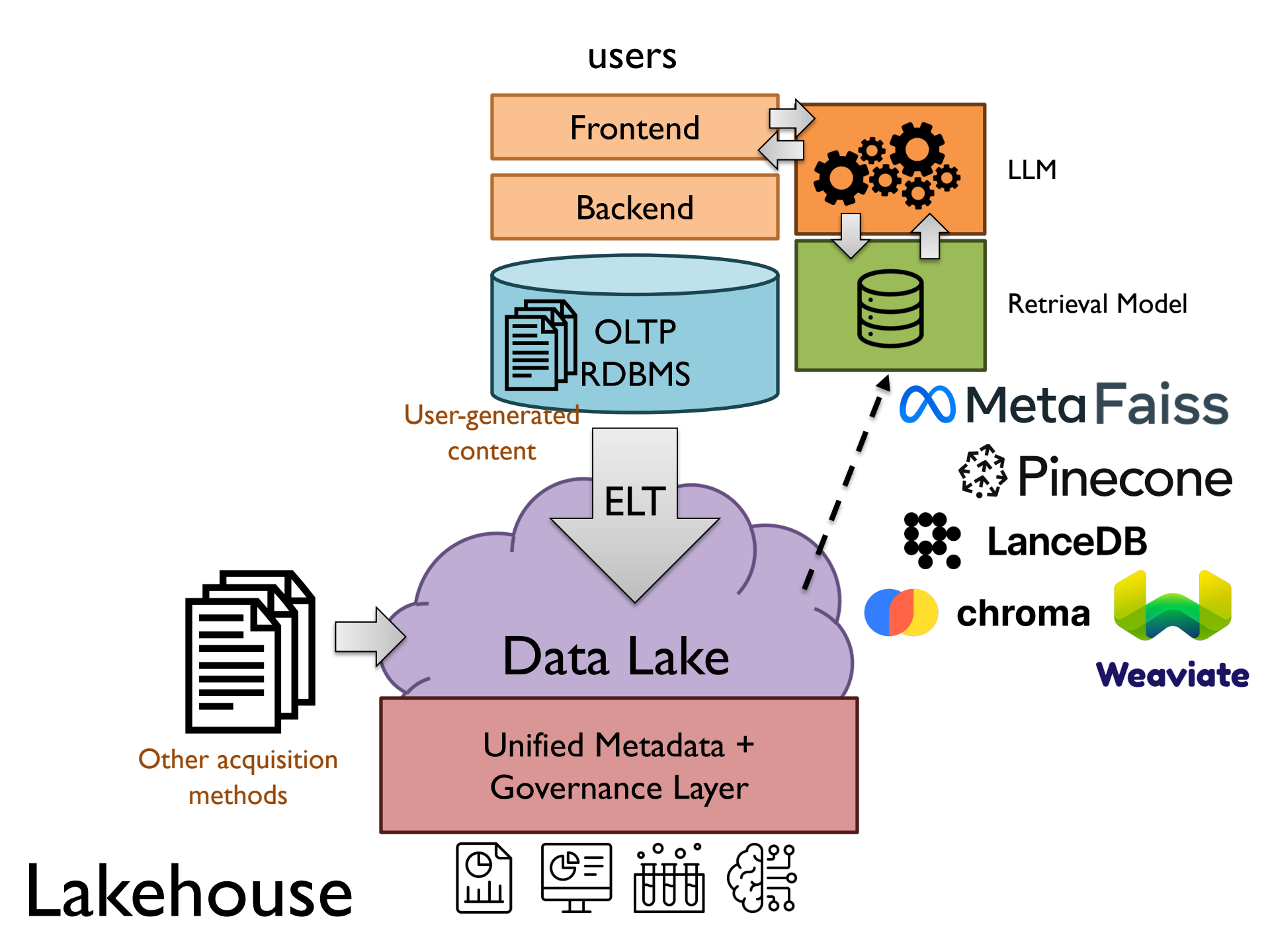
Other sources…

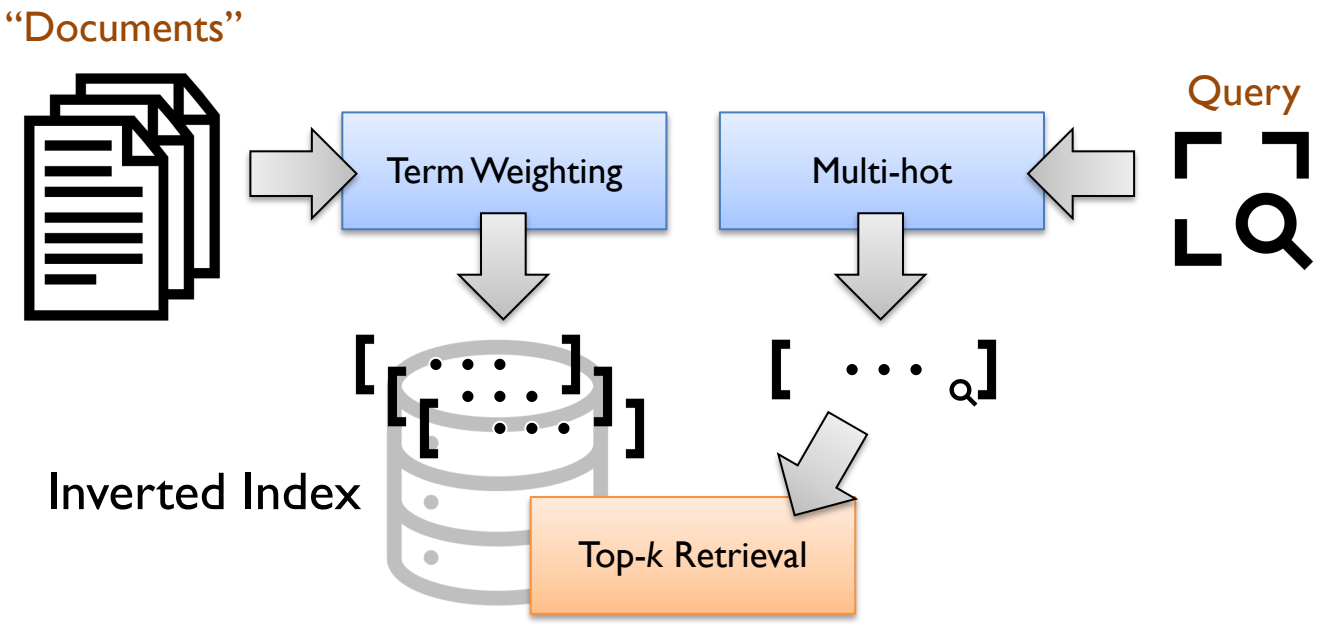
Performing "standard" lakehouse tasks

Joining, filtering, projecting, etc. heterogenous data

Data cleaning, aggregation, etc.

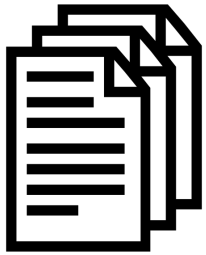
Training ML models


users
Frontend
Backend
LLM
Retrieval Model
OLTP
RDBMS
User-generated
content
ELT
Meta Faiss
Pinecone
LanceDB
chroma
Weaviate
Data Lake
Unified Metadata +
Governance Layer
Other acquisition
methods
Lakehouse

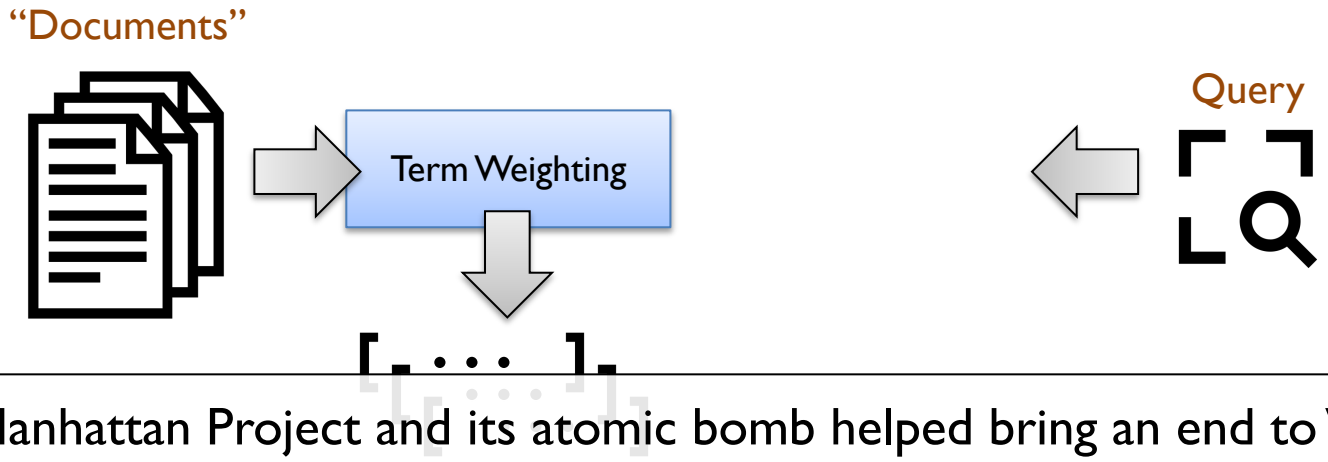


Challenge #1: How do we assign weights?

Challenge #2: How do we perform top-*k* retrieval efficiently?



Results

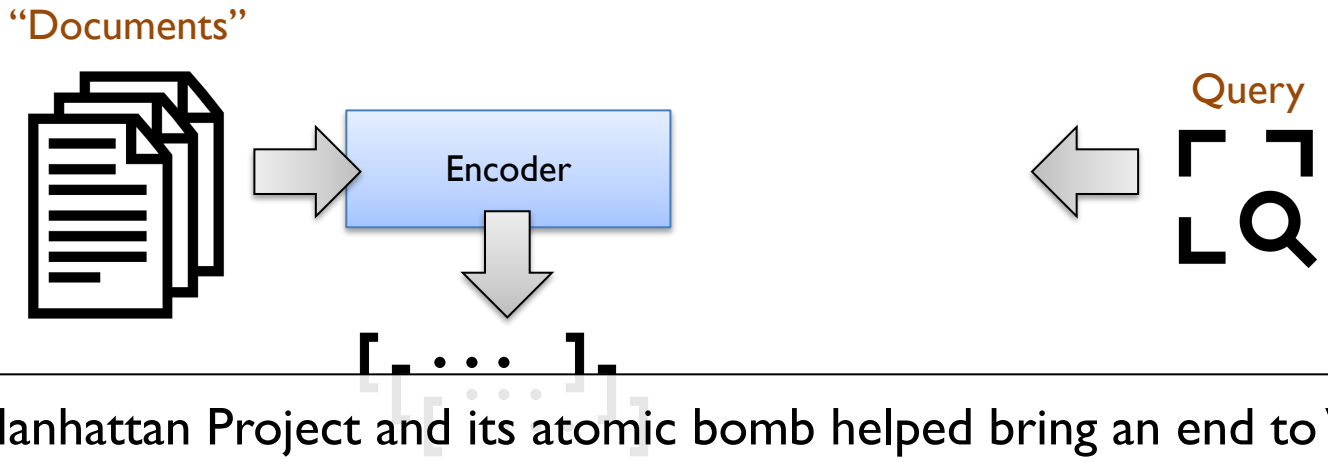


The Manhattan Project and its atomic bomb helped bring an end to World War II. Its legacy of peaceful uses of atomic energy continues to have an impact on history and science.

```
{'atom': 4.0140, 'bomb': 4.0704, 'bring': 2.7239, 'continu':
2.4331, 'end': 2.1559, 'energi': 2.5045, 'have': 1.0742, 'help':
1.8157, 'histori': 2.4213, 'ii': 3.0998, 'impact': 3.0304, 'it':
2.0473, 'legaci': 4.1335, 'manhattan': 4.1345, 'peac': 3.5205,
'project': 2.6442, 'scienc': 2.8700, 'us': 0.9967, 'war':
2.6454, 'world': 1.9974}
```

sparse vector

Results



The Manhattan Project and its atomic bomb helped bring an end to World War II. Its legacy of peaceful uses of atomic energy continues to have an impact on history and science.

```
[0.099843978881836, 0.870057582855224 6, 0.520509719848633,
0.030491352081299, 0.723929882049560 5, 0.134523391723633,
0.433127403259277 3, 0.644286632537842, 0.645430564880371,
0.047342777252197 3, 0.070496082305908, 0.504533529281616,
0.815732955932617 2, 0.133575916290283, 0.997444868087768 6,
0.074254274368286 1, 0.155941247940063 5, 0.421395778656006,
0.014032363891602, 0.996794581413269...]
```

dense vector

Results

Challenge #2: Efficient Top-*k* Retrieval

Do you need both?

Sparse vectors "text databases"

Dense vectors vector databases

Challenge #2: Efficient Top-*k* Retrieval

Do you need both?

Yes, hybrid fusion – combining results of both – often outperforms individual results

Min/Max normalization + Score averaging

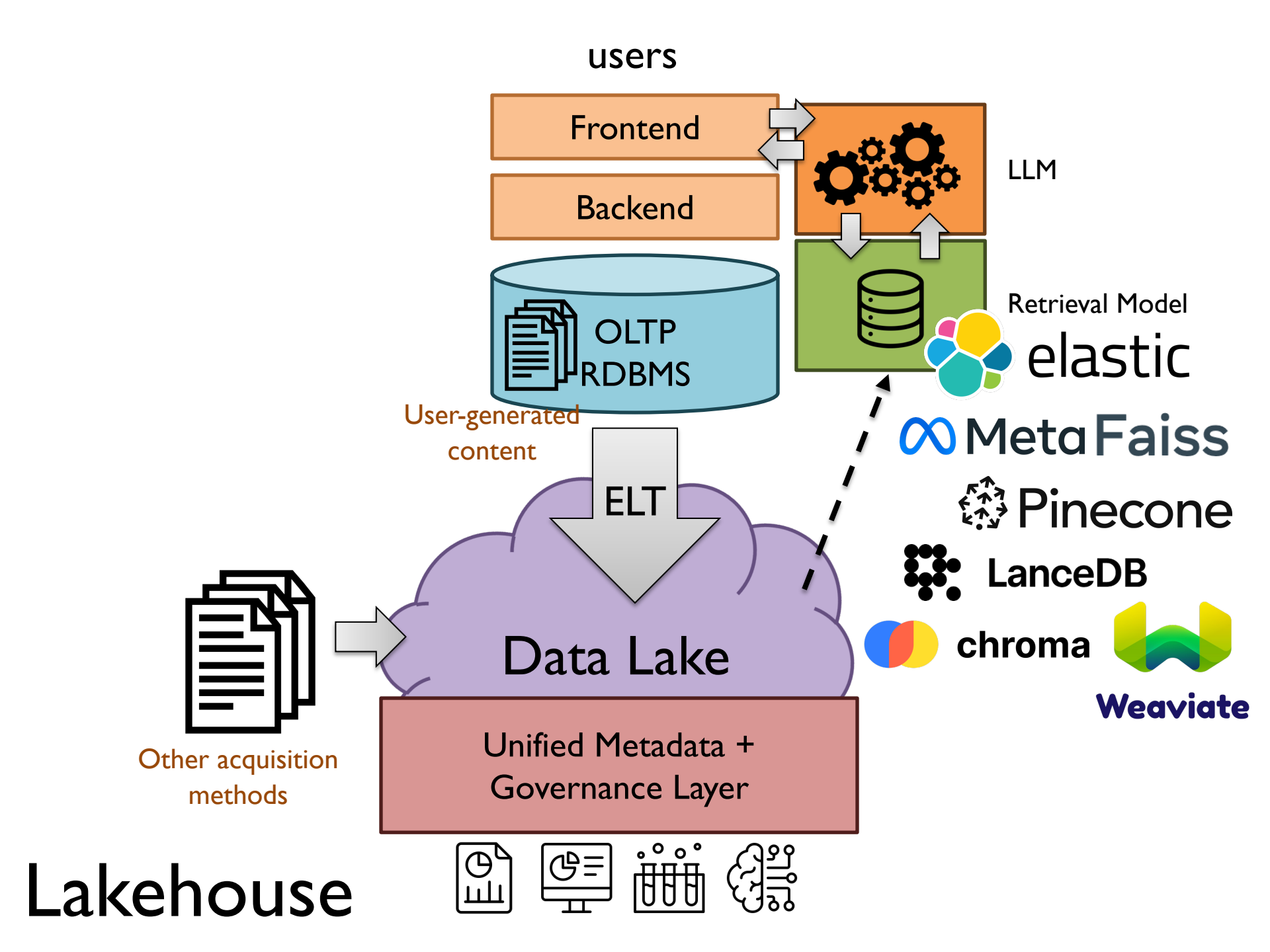
Reciprocal Rank Fusion

RRF simply sorts the documents according to a naive scoring formula. Given a set D of documents to be ranked and a set of rankings R, each a permutation on $1..|D|$, we compute

$$RRFscore(d \in D) = \sum_{r \in R} \frac{1}{k + r(d)},$$

where $k = 60$ was fixed during a pilot investigation and not altered during subsequent validation. Our intuition in

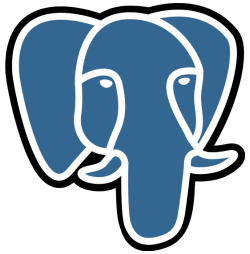
Cormack et al. Reciprocal Rank Fusion outperforms Condorcet and individual Rank Learning Methods. *SIGIR 2009*.


users
Frontend
Backend
LLM
OLTP
RDBMS
Retrieval Model
elastic
User-generated
content
ELT
Meta Faiss
Pinecone
LanceDB
chroma
Weaviate
Data Lake
Other acquisition
methods
Unified Metadata +
Governance Layer
Lakehouse

Revenge of the DBs?

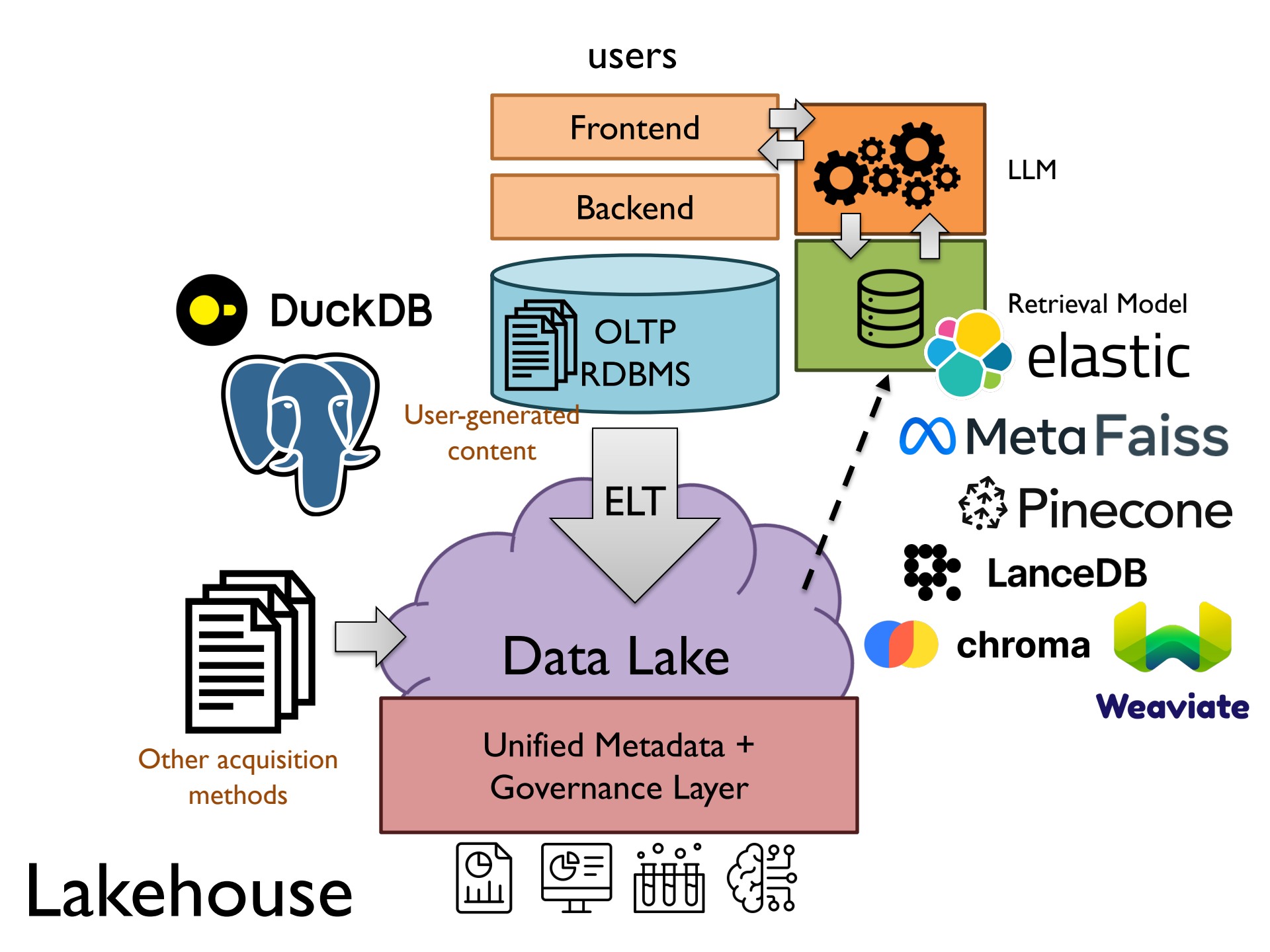
DuckDB supports full-text search via the fts extension
https://duckdb.org/docs/stable/guides/sql_features/full_text_search

The vss extension is a DuckDB extension to support vector similarity search
https://duckdb.org/docs/stable/core_extensions/vss



pg_search: adds functions/operators for efficient search
https://neon.com/docs/extensions/pg_search

pg_vector: open-source vector similarity search
https://github.com/pgvector/pgvector


users
Frontend
Backend
LLM
DuckDB
OLTP
RDBMS
Retrieval Model
elastic
User-generated
content
Meta Faiss
ELT
Pinecone
LanceDB
chroma
Weaviate
Data Lake
Other acquisition
methods
Unified Metadata +
Governance Layer
Lakehouse


富嶽三十六景
神奈川沖
浪裏