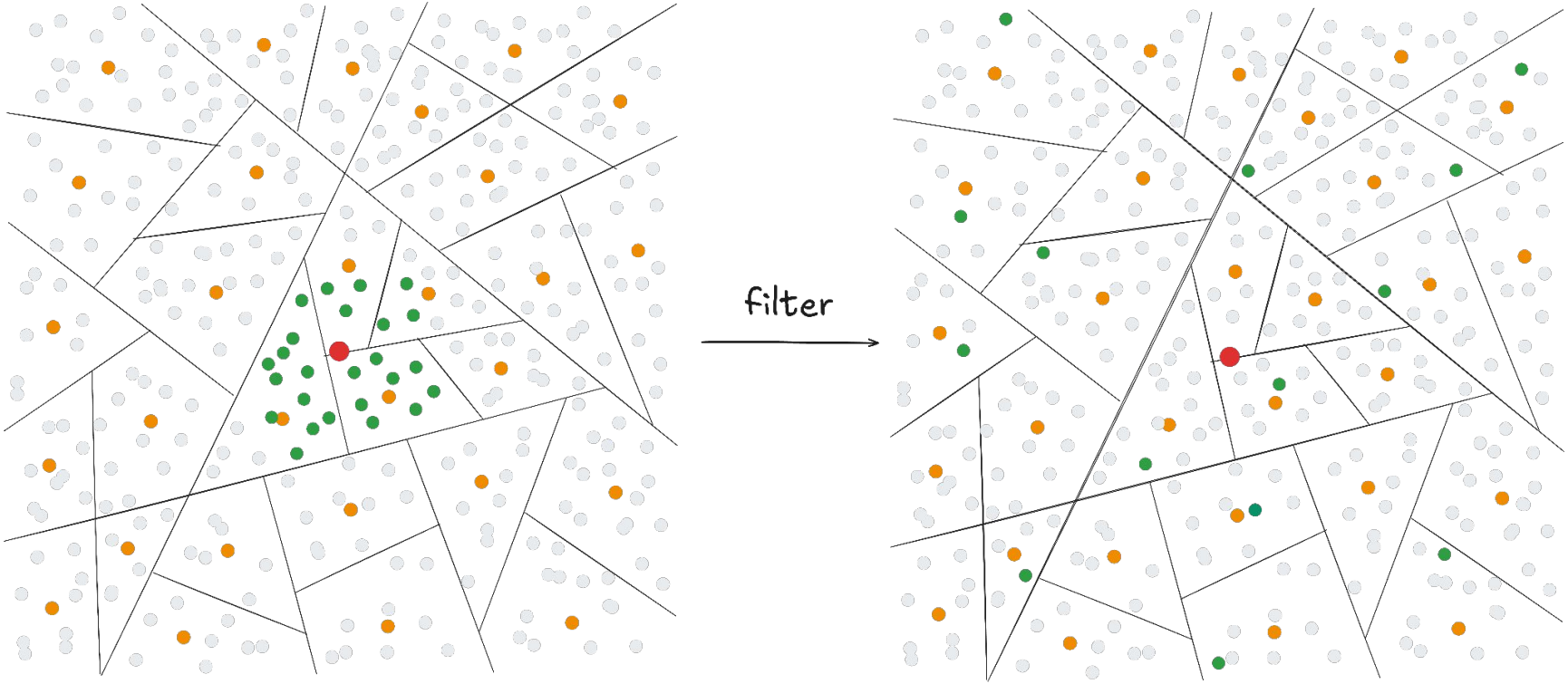
Billion-scale hybrid retrieval in a single query

whoami

- Co-Founder & CEO @ TopK
- ex-Pinecone engineering lead (data plane & control plane)
- ex-Shopify risk algorithms (fraud detection & forecasting)
- Game theory and adversarial ML research @ CTU Prague

Vector databases are the wrong abstraction

Uncorrelated vector & metadata distributions



Inflexible scoring

- Score engineering at query time is important to optimize relevance
  - Keyword boosting
  - Prioritizing more recent documents
  - Boosting “nearby” results (geosearch)
  - …
- Vector database indexes approximate a specific score (e.g. cosine similarity)
- Modifying the scoring objective at query time is impossible without rebuilding the entire index (slow and expensive)

Expensive writes

- Writes or deletes introduce data distribution shift
- Vector indexes need to be periodically rebuilt on new data to maintain recall
- Indexing throughput is negatively impacted due to frequent rebuilds

Operational complexity

- Coupled compute-storage
  - Requires replication for durability (networking cost)
  - Hard to optimize for storage heavy and query heavy workloads
- Coupled write-read path
  - Unpredictable tail latencies
  - Difficult to scale writes and reads separately

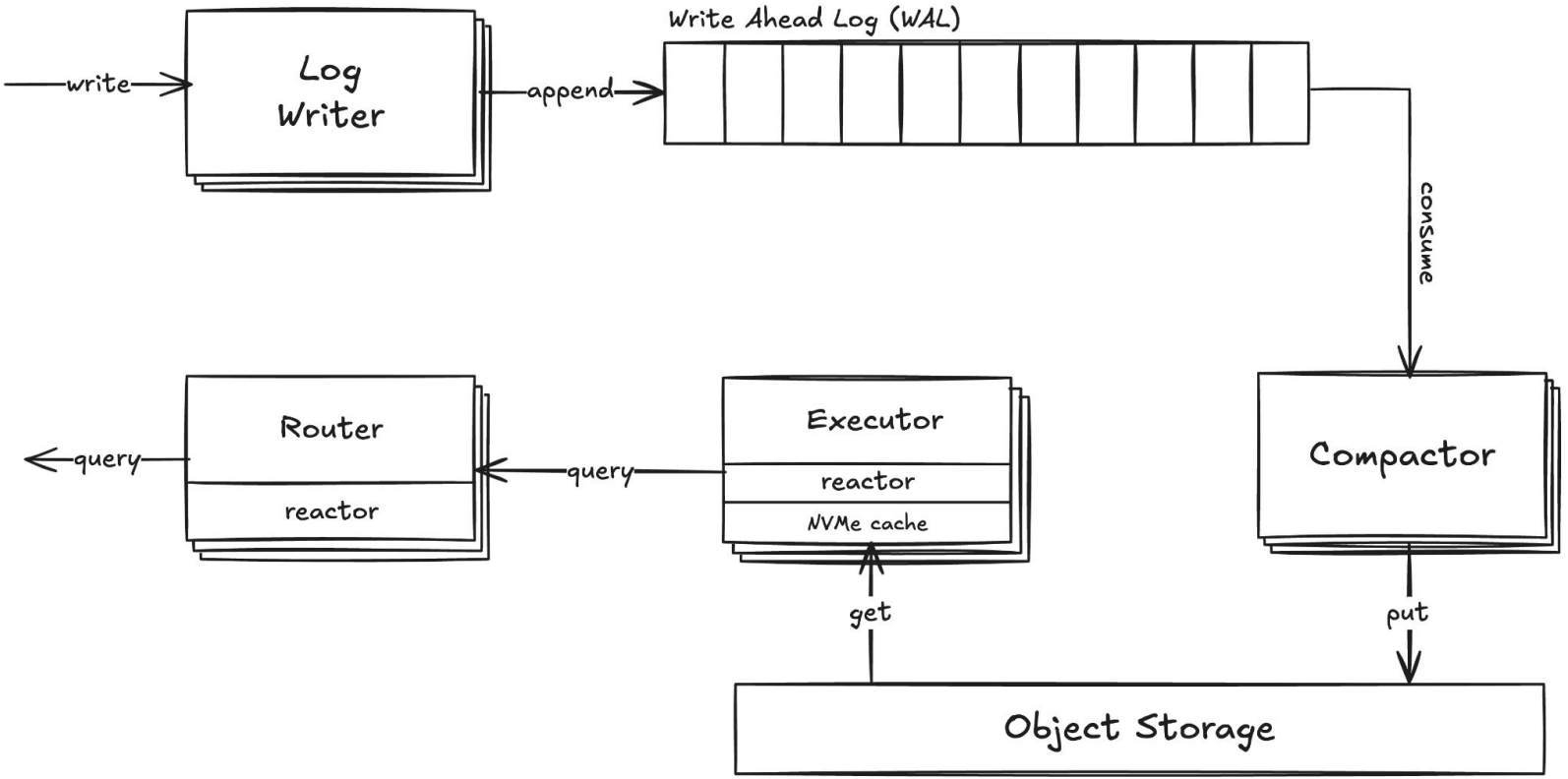
What we built instead

Unified retrieval engine

- **Custom query engine (*reactor*)**
  - Dense and sparse vectors (f32, u8, i8, binary)
  - Supports multi-vector retrieval
  - Lexical search with BM25 scoring
  - Custom scoring at query time
  - Efficient filtering that preserves results quality
- **Fully-managed cloud-native search database**
  - Separate compute and storage
  - Separate read path and write path
  - Storage format optimized for object storage as the primary medium

Least common denominator dependencies

- **Object storage (S3, …)**
  - Our primary and only durable storage
  - 99.99% availability, 11 9s of durability
  - Cheap ~0.023 $/GB
  - Virtually infinite capacity and r/w throughput
  - Support for conditional writes (CAS semantics)
- **Compute (EC2, …)**
  - Local NVMe SSDs with >GB/s throughput and sub-1ms I/O latency
  - High bandwidth networking
  - ARM >> x86 (with custom NEON kernels), DDR5 memory
  - Abstracted through k8s


write
Log Writer
append
Write Ahead Log (WAL)
consume
Router
reactor
query
query
Executor
reactor
NVMe cache
Compactor
get
put
Object Storage

Write path

- **WAL (Write-Ahead Log)**
  - Leverages object storage with CAS (S3, GCS, Azure Blob, …)
  - Linearizable semantics => strong consistency
  - No external dependencies for sequencing
  - Achieves ~80MB/s throughput per collection

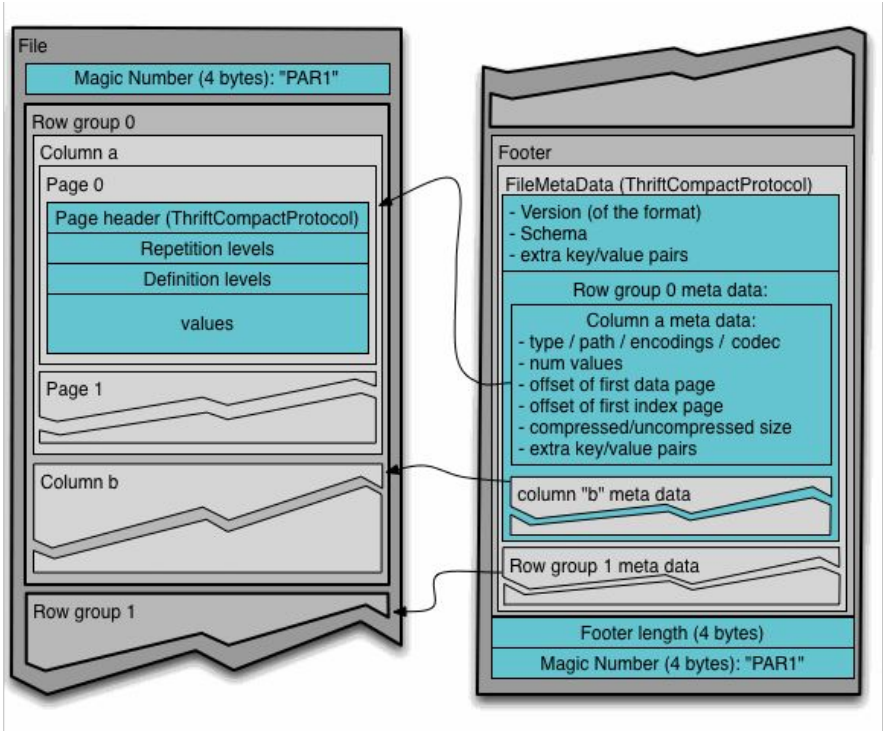
- **Compaction**
  - Produces read-optimized version of WAL segments
  - Garbage collection of deleted & updated records
  - Garbage collection of old data files and log files

.bob

(just a bunch of buffers)

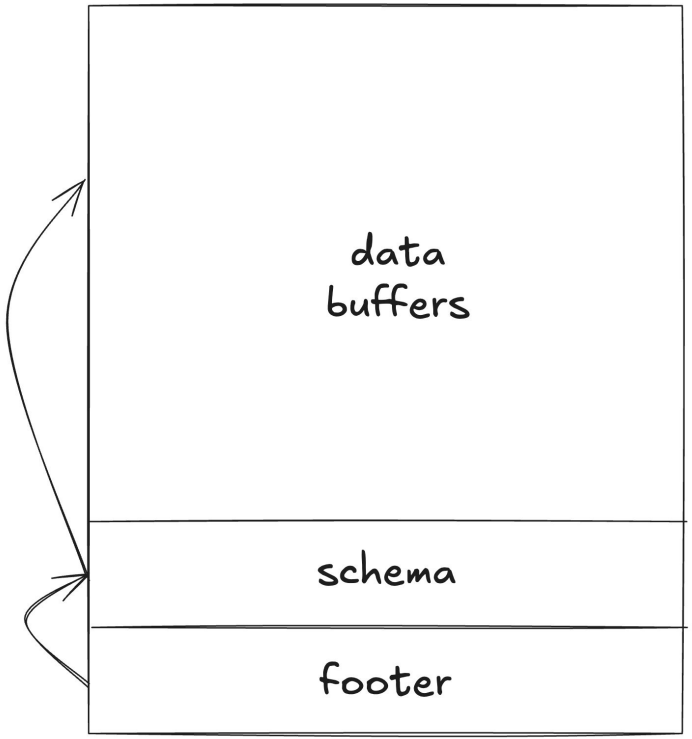
Problems with *.parquet*

- Built for spinning disks (lots of serial dependencies in I/O)
- Couples I/O granularity with metadata (stats) granularity
- No support for point reads
- Requires decoding full metadata to access just one column
- Uses off-the-shelf compression (e.g. snappy, zstd)



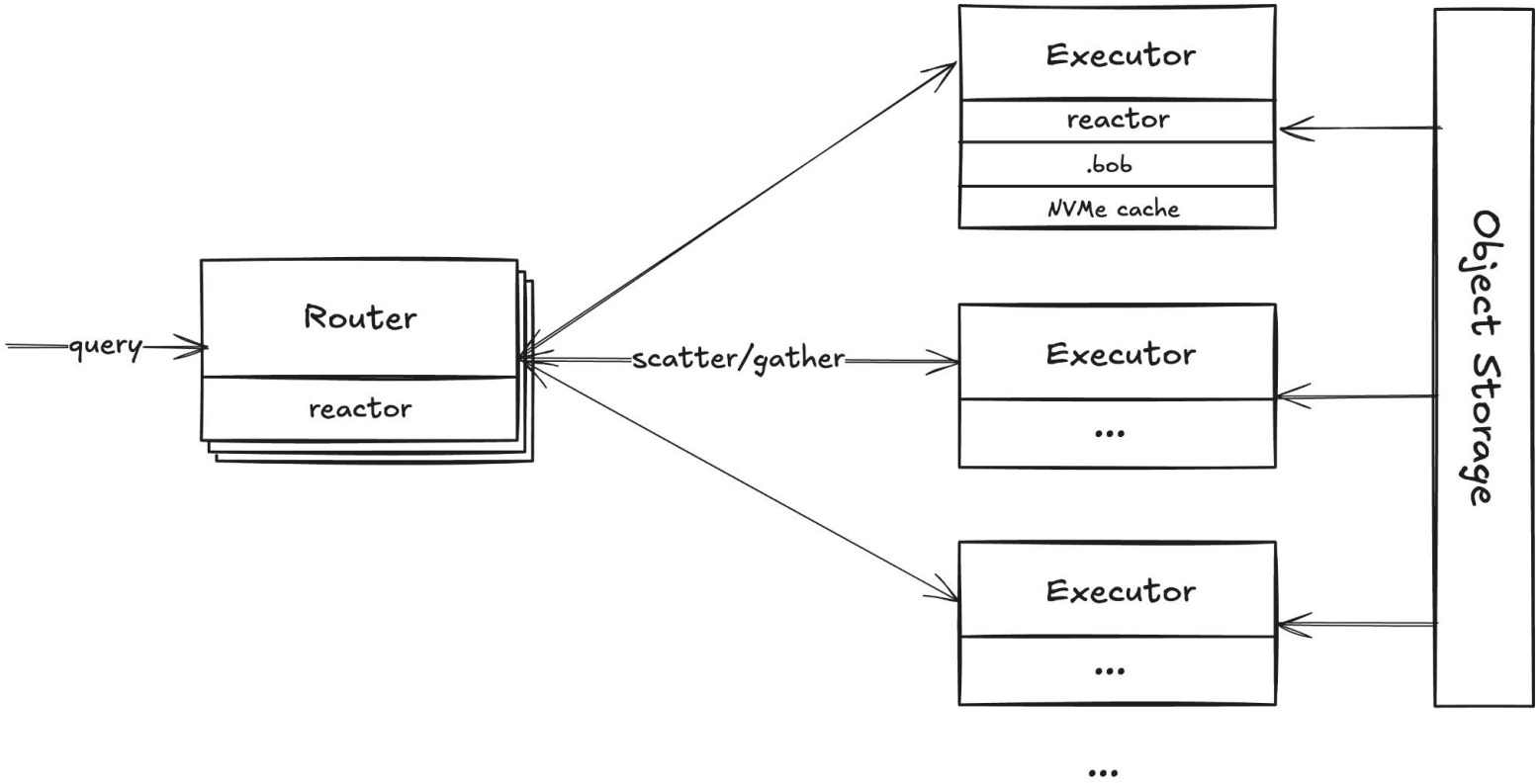
.bob columnar layout

- Decouples logical & physical file structure.
- **Physically**, all data is stored as buffers.
- **Logically**, rows in the file are organized as columns (per field) split into blocks.
- Achieves optimal I/O size and stats granularity for pruning.
- Uses Apache Arrow and type-specific compression for zero-copy/decode execution.
- Supports point reads



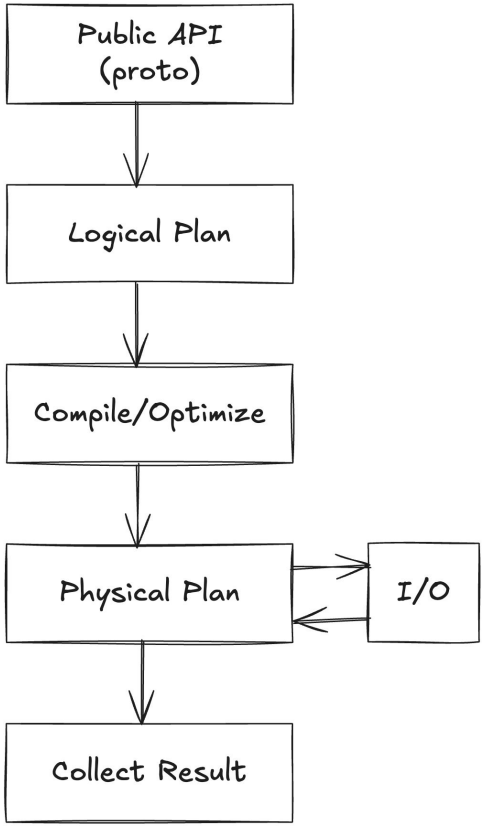
reactor

(where datafusion happens)


Executor
reactor
.bob
NVMe cache
Object Storage
query
Router
reactor
scatter/gather
Executor
...
Executor
...
...

Query lifecycle

- User queries are parsed to *logical plan* that describes **what** should be computed
- Logical plans are **compiled** and **optimized** against a specific schema.
- Resulting physical plan describes **how** the computation happens.
- Physical nodes can read local or remote data streamed from other nodes (e.g. executor -> router).



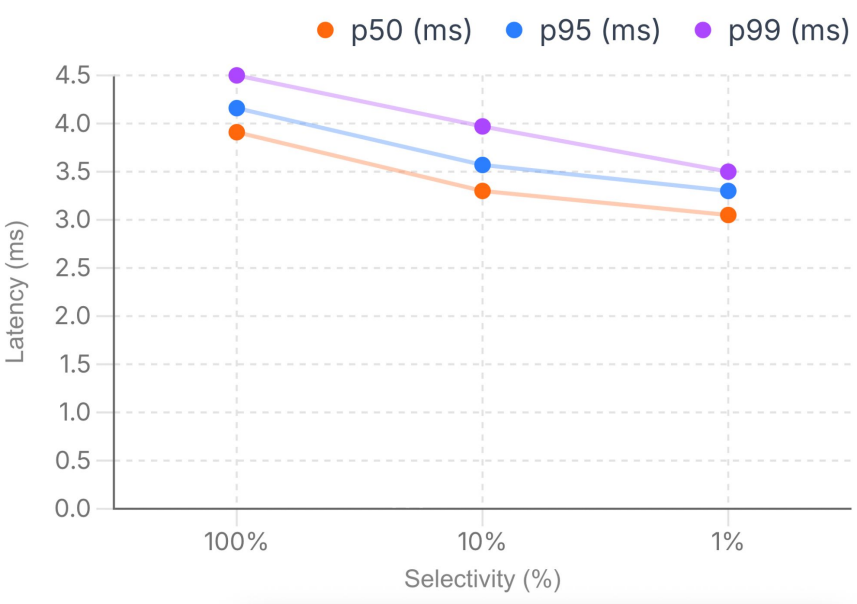
Features

- Works with dynamic schemas (TopK has schema-on-write model)
- Uses Apache Arrow as in-memory format
- Native support for external indexes (no need to join)
- Projection pushdown
- Predicate pushdown with block pruning
- Zero-copy execution for buffers already cached in memory
- Distributed execution with O(100) nodes

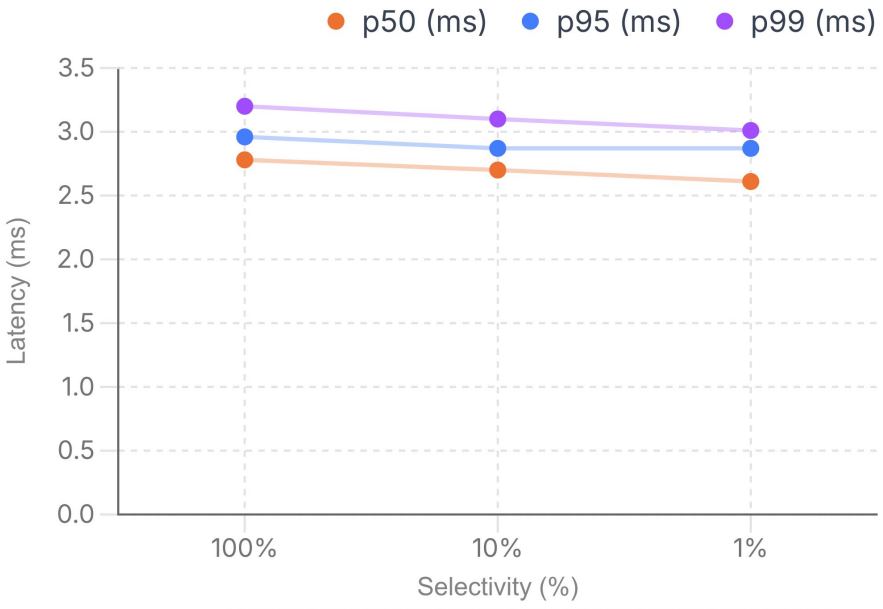
But is it fast?

1m documents

dense

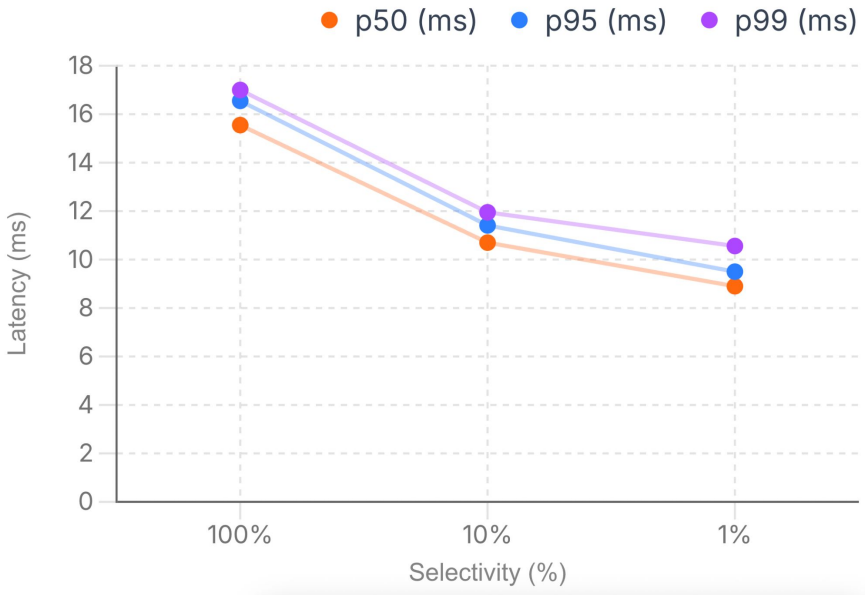


sparse

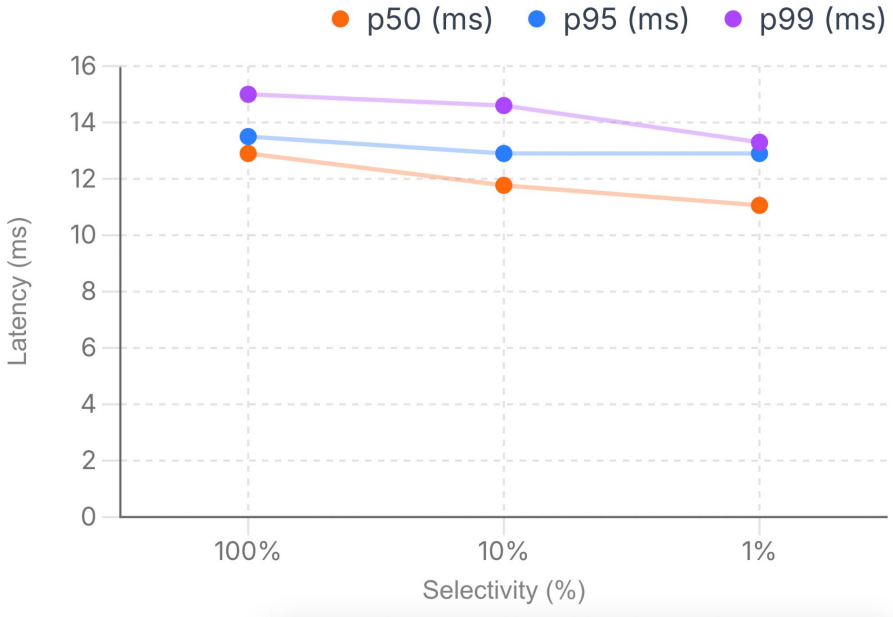


10m documents

dense

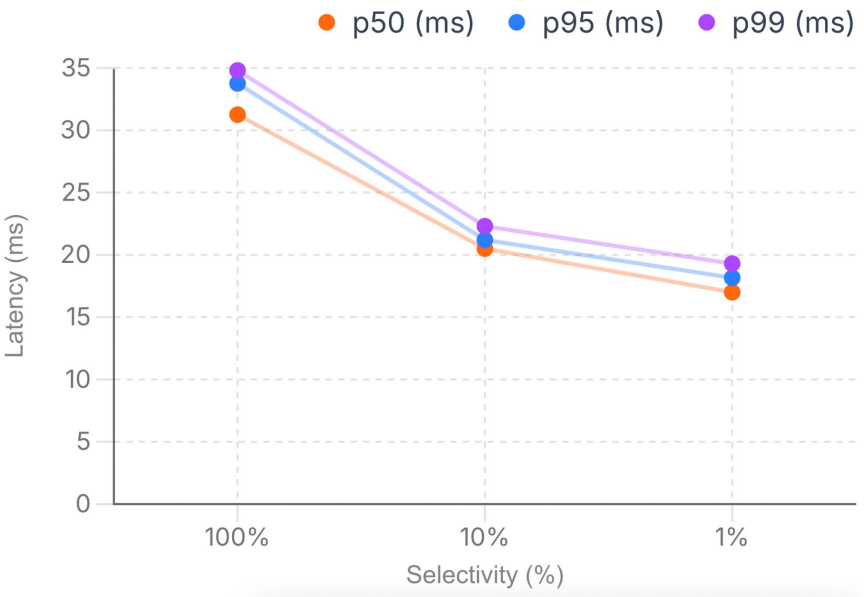


sparse

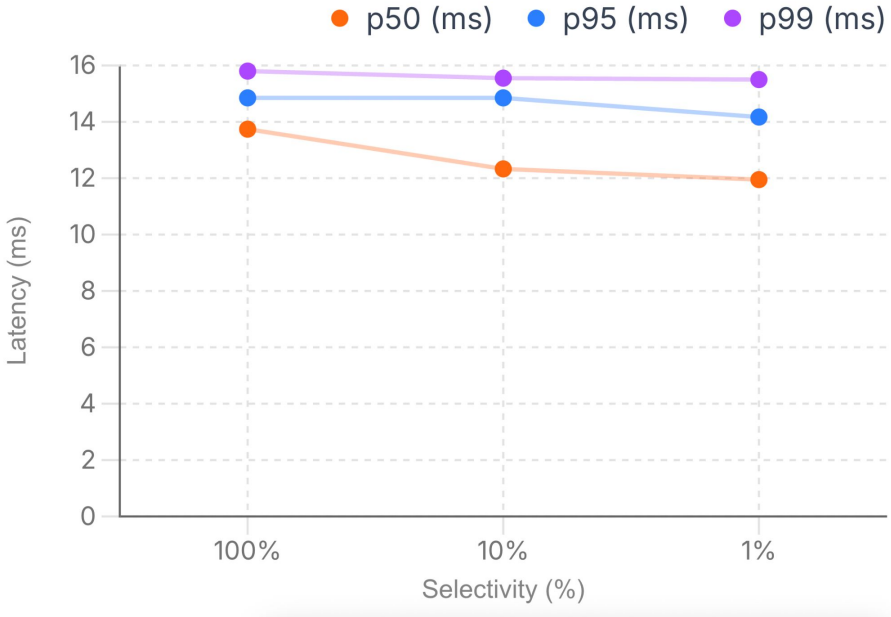


100m documents

dense

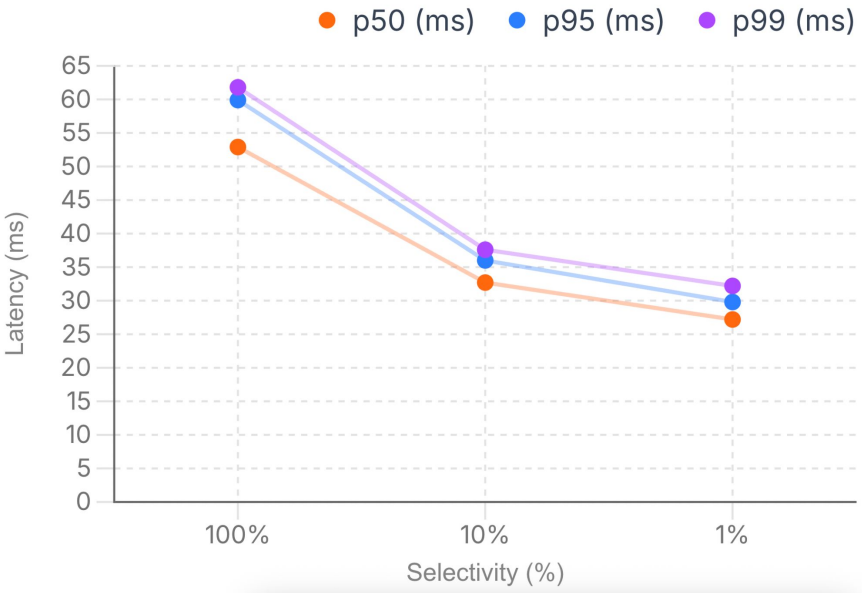


sparse

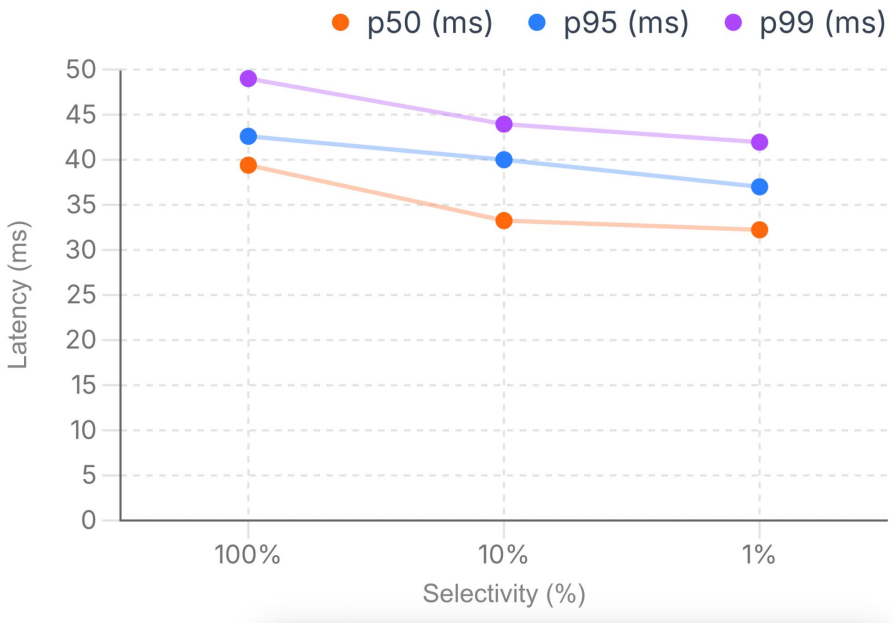


1B documents

dense

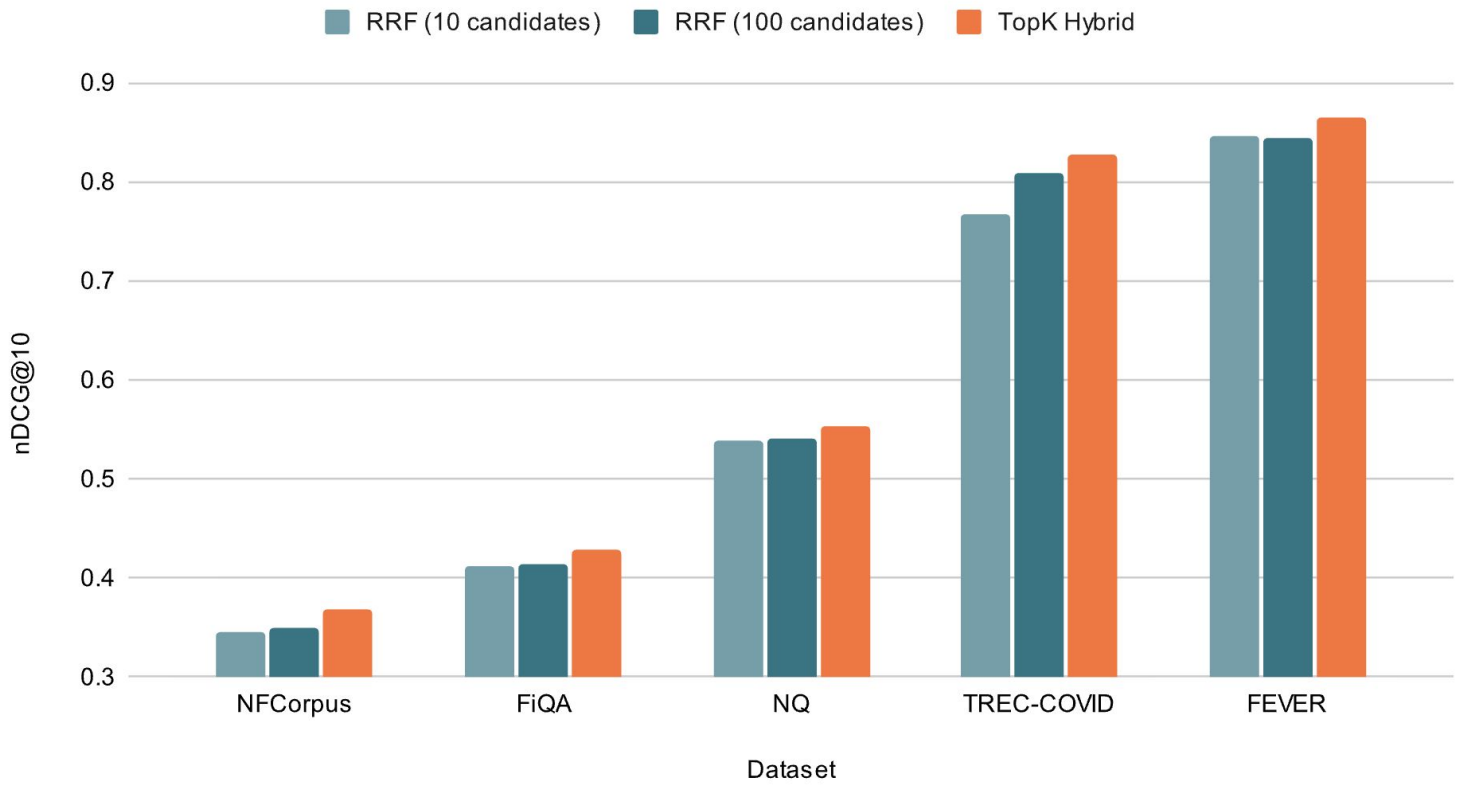


sparse



How about quality?

RRF vs. TopK Hybrid



What's next?

Search is changing

Q&A

QUESTION

What is the capital of France?

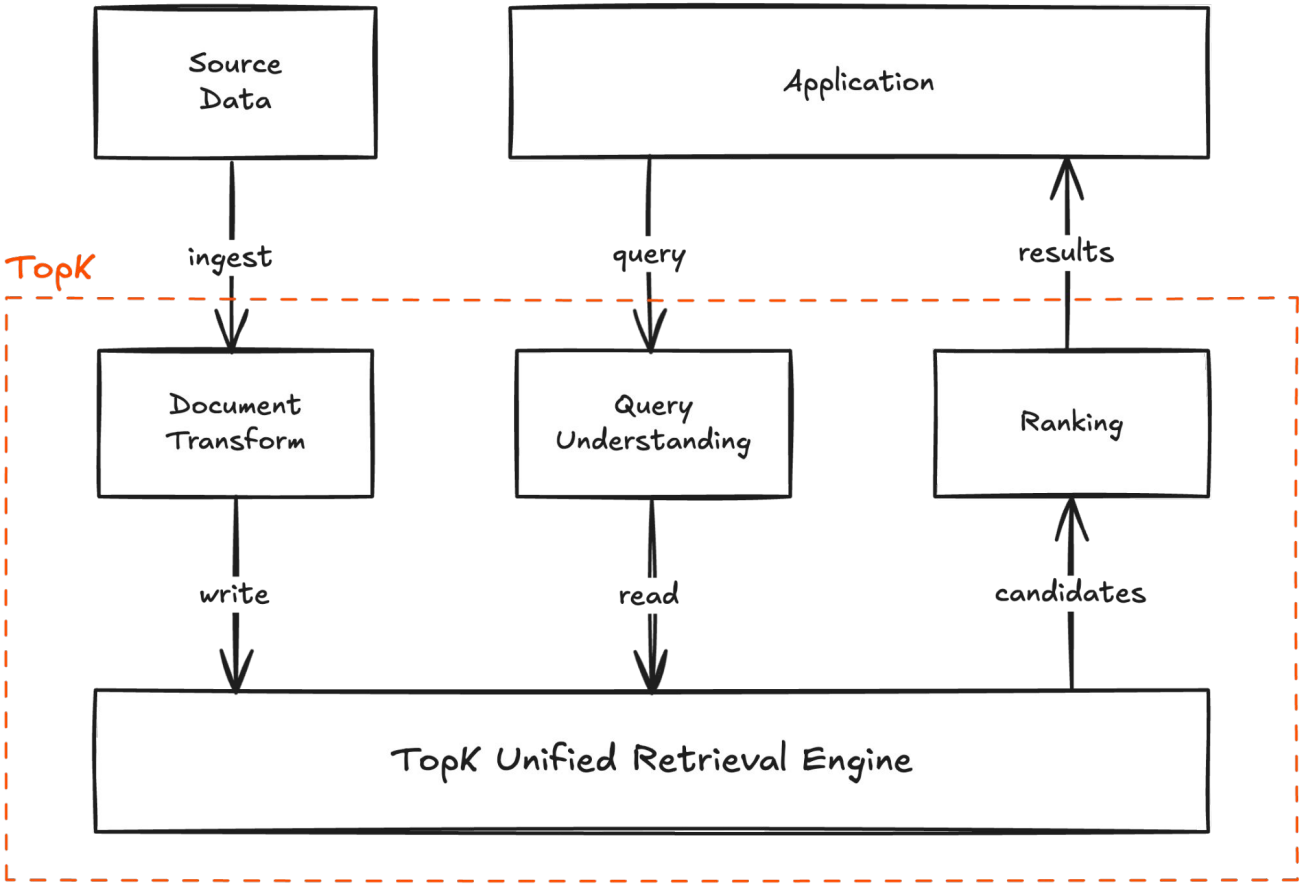
ANSWER

The capital of France is Paris. It has served as the political, cultural, and economic center of the country for centuries and is one of the most visited cities in the world.

CITATIONS

Britannica – Paris

CIA Wold Factbook –


Source Data
Application
TopK
ingest
query
results
Document Transform
Query Understanding
Ranking
write
read
candidates
TopK Unified Retrieval Engine

Thanks

topk.io