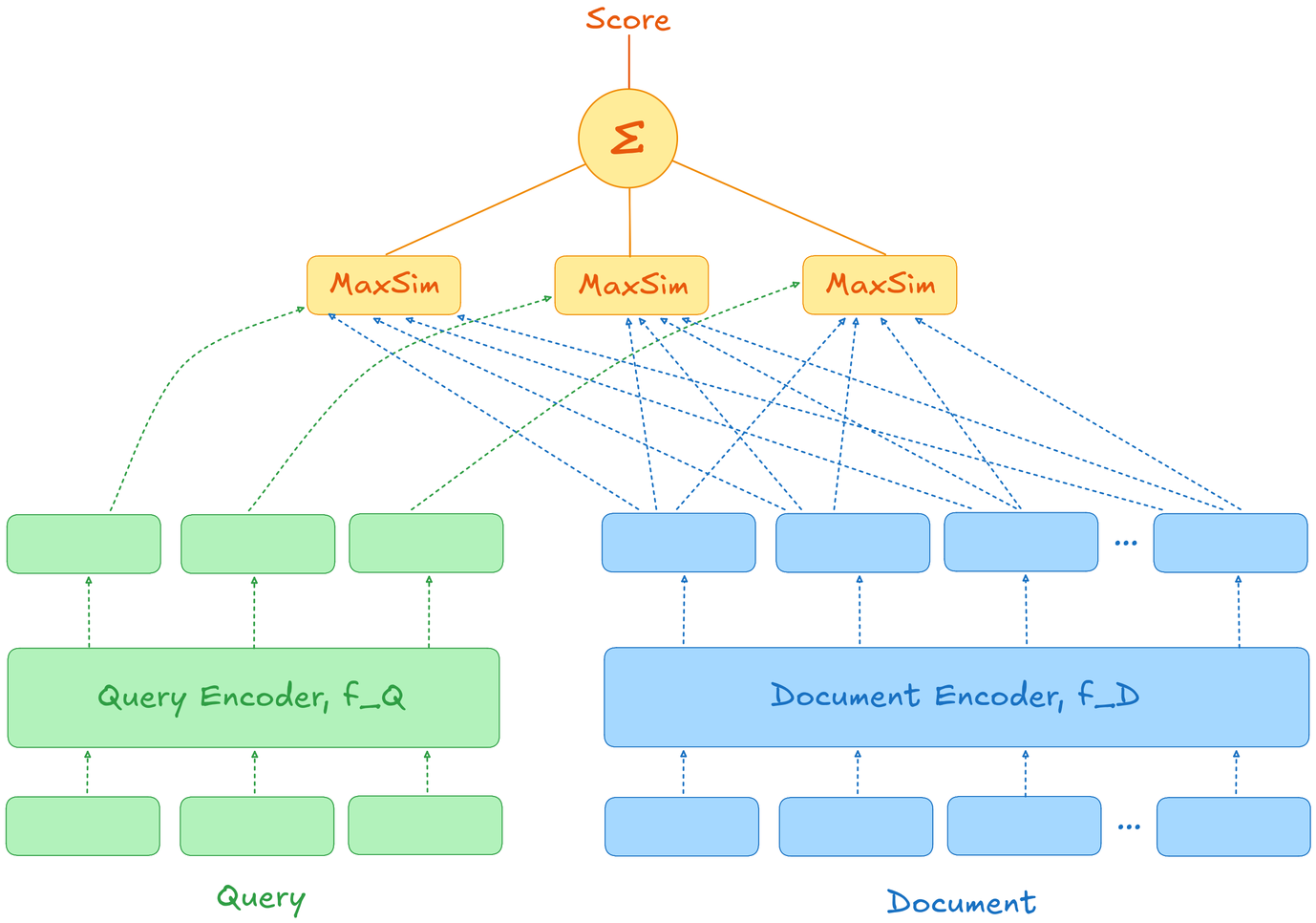
Late interaction beyond ColBERT

George Panchuk
HSE AI Fall 2025

github.com/joein/vector-search-course

ColBERT recap

- Late interaction is a retrieval paradigm where queries and documents are encoded independently, but their token-level interactions are computed only at search time to recover fine-grained relevance signals.
- ColBERT is the first model to publicly formalise Late Interaction and demonstrate that it can be used in ANN
- Relevance is computed using *MaxSim operator*, matching each query token to its most similar document token
- ColBERT produces a 128 dim vector for each of the tokens
- Might provide a better recall than representation based models, as well as be faster than full interaction models like cross-encoders
- Though, the authors claims that ColBERT can be a first-stage retrieval, the actual usage might be tricky and expensive
- The authors propose a multistage IVFPQ pipeline that first prefetches candidates by retrieving documents whose token embeddings are closest to each query token, and then reranks this candidate set using the original vectors.
- Despite ANN algorithms can operate on ColBERT vectors, the sheer number of token-level embeddings makes index construction computationally intensive, regardless of the underlying ANN algorithm.

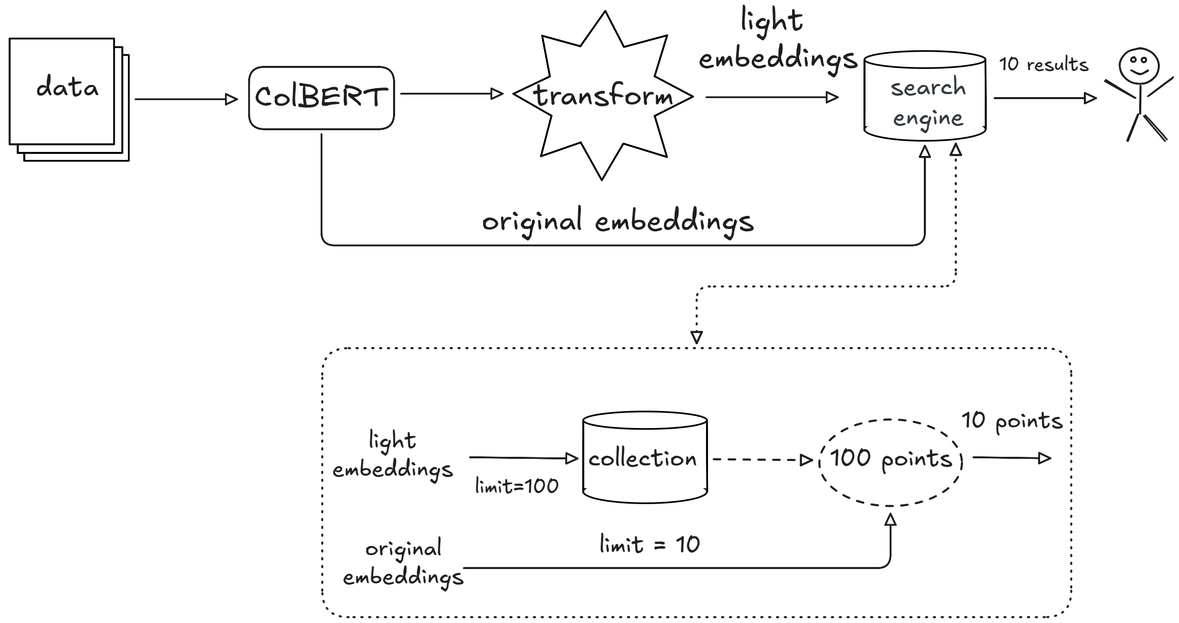


ColBERT is slow for retrieval

How to actually search with late interaction?

- As the authors proposed, the practical way for retrieving with ColBERT is to build a multistage pipeline
- The first stage prefetches a large set of candidates, whilst ColBERT narrows it down
- There are several ways to implement the first stage, some of them are:

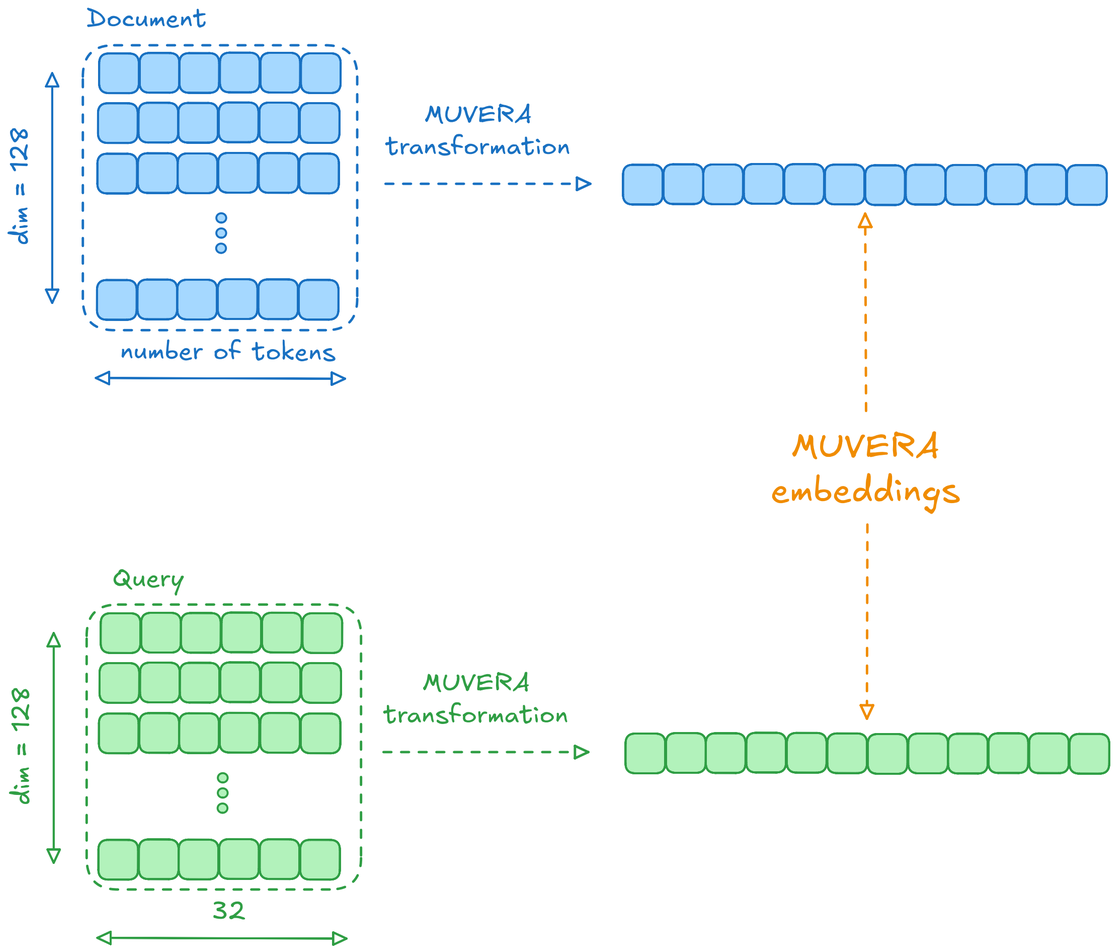
  1) use a different model to select a pool of candidates
  2) create heuristics like looking for the documents closest to each of the query tokens independently
  3) use ColBERT vectors transformation methods



MUVERA

Multivector Retrieval Algorithm

- Goal: Compress a multi-vector representation into a single fixed-dimensional encoding (FDE).
- Document and queries contain variable numbers of token embeddings, so we need a uniform format.
- How it works:
  - Partitions the token-embedding space into a fixed number of regions via SimHash (LSH with random hyperplanes)
  - Aggregates token vectors within each region, producing a structured, fixed-layout representation
  - Concatenating outputs from multiple independent projections, forming a rich fixed dimensional encoding (FDE).
- SimHash assigns each token vector to one of 2^{k_sim} clusters using only hyperplane sign tests, enabling fast, consistent clustering across all documents and queries.



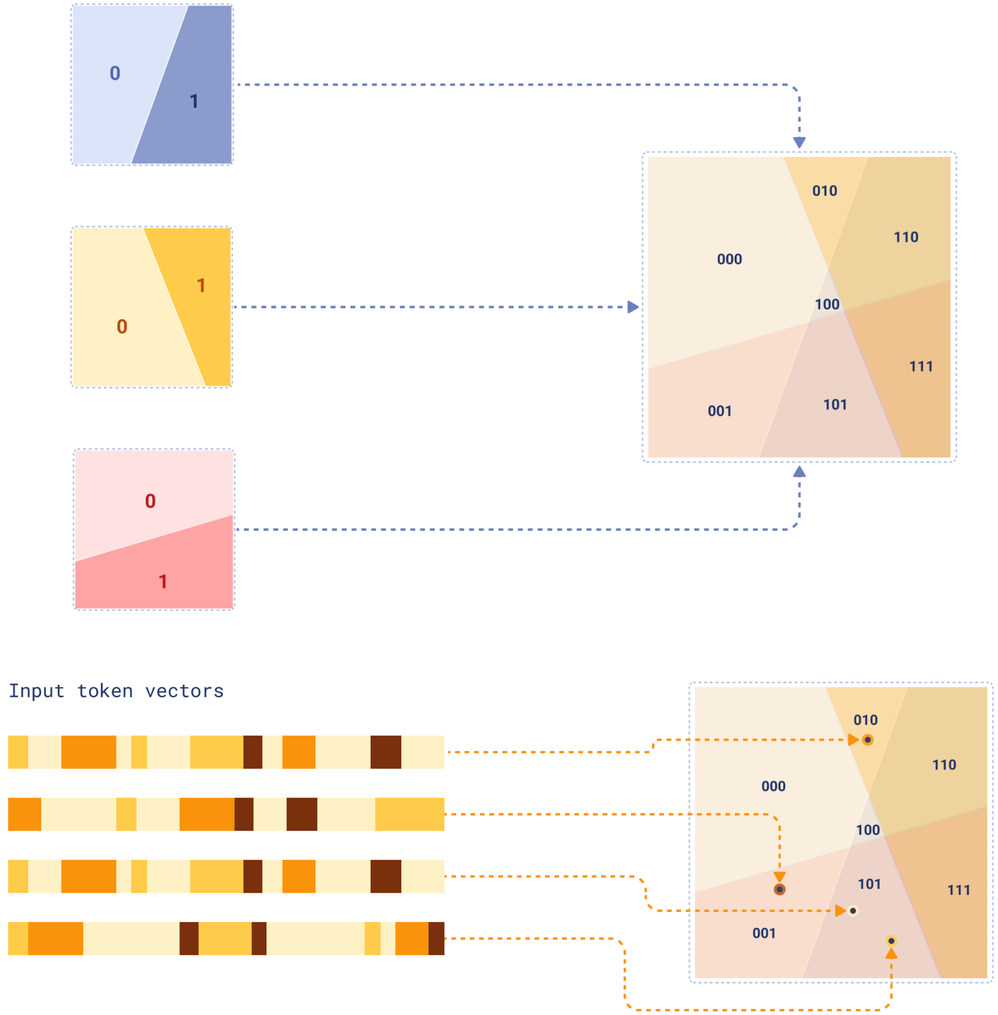
MUVERA

SimHash recap

SimHash is an LSH technique that maps vectors to binary codes using random hyperplanes

How it works:

- Sample k_{sim} random hyperplanes (normal vectors drawn from a standard normal distribution)
- Each hyperplane splits the space into 2 halves, together forming 2^{k_sim} regions
- For a token vector x, compute $sign(x \cdot h_i)$ for every hyperplane h_i
- The sequence of signs forms a k_{sim}-bit binary code
- Interpret this binary code as an integer → this becomes the cluster ID.



MUVERA: Making multi-vectors more performant

MUVERA

Fixed Dimensional Encoding

Once token vectors are assigned to SimHash clusters, MUVERA aggregates them differently for documents and queries.

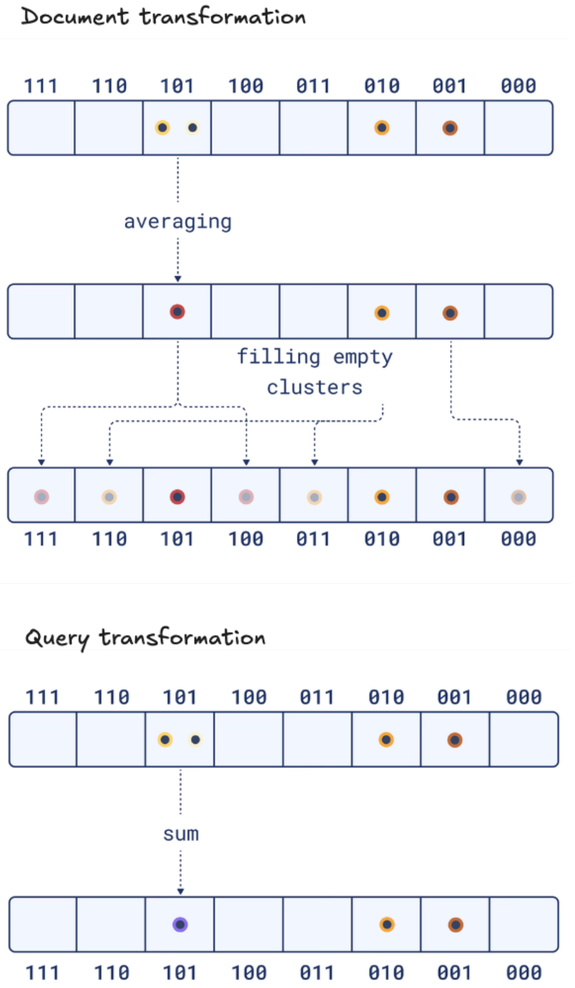
Document aggregation:

- For each cluster, compute the centroid (mean) of all assigned vectors
- If a cluster is empty, fill it using the a vector from the nearest non-empty cluster (not centroid) (nearest = smallest Hamming distance between cluster IDs)

Output: 2^{k_sim} vectors, each of dimension *dim*

Query aggregation:

- For each cluster, compute the sum of assigned vectors (not the mean). Larger cluster → larger magnitudes → reflects query term frequency.
- Do not fill empty clusters: Queries are short → empty cluster are expected. Filling them would add artificial terms and distort the scoring.

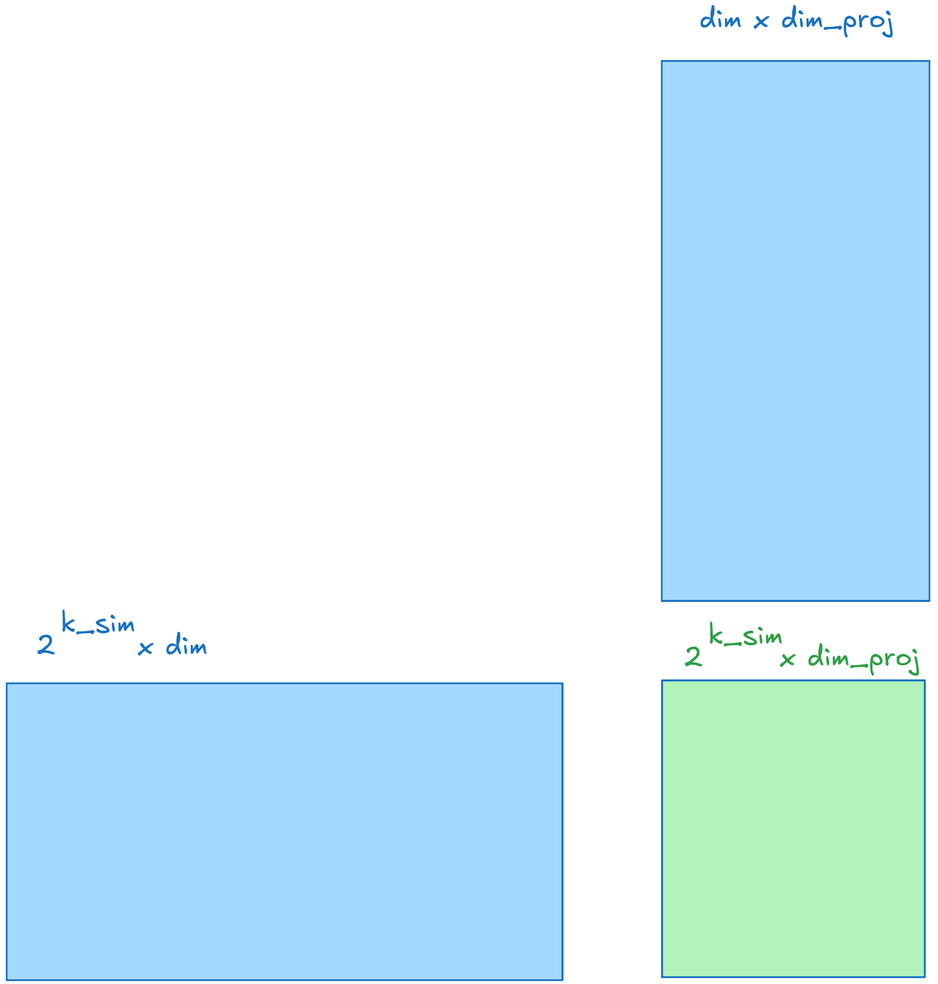


MUVERA: Making multi-vectors more performant

MUVERA

Dimensionality reduction through Random Projection

- With small SimHash codes (k_{sim}=4 or 5 - we get only 16 or 32 clusters), a single projection may be too coarse. To increase expressiveness, MUVERA repeats the clustering-and-aggregation process r_reps times, each with independent random hyperplanes.
- Example (dim=128, k_{sim} =4): $FDE = 16 \cdot 128 = 2048$; After r_reps: $20 \cdot 2048 = 40960$
- Sizes like 40960 are probably too large for efficient ANN search
- To keep the FDE compact, each cluster vector is projected using a random {-1, +1} matrix of shape (dim, dim_{proj}) and applying a scaling factor of $\frac{1}{\sqrt{dim_{proj}}}$
- The resulting FDE dim_{fde} is then $r\_reps \cdot 2^{k\_sim} \cdot dim_{proj}$
- The original paper suggests applying an additional random projection to the final FDE to further reduce its dimensionality, however, it's an optional step.



MUVERA

Benchmark

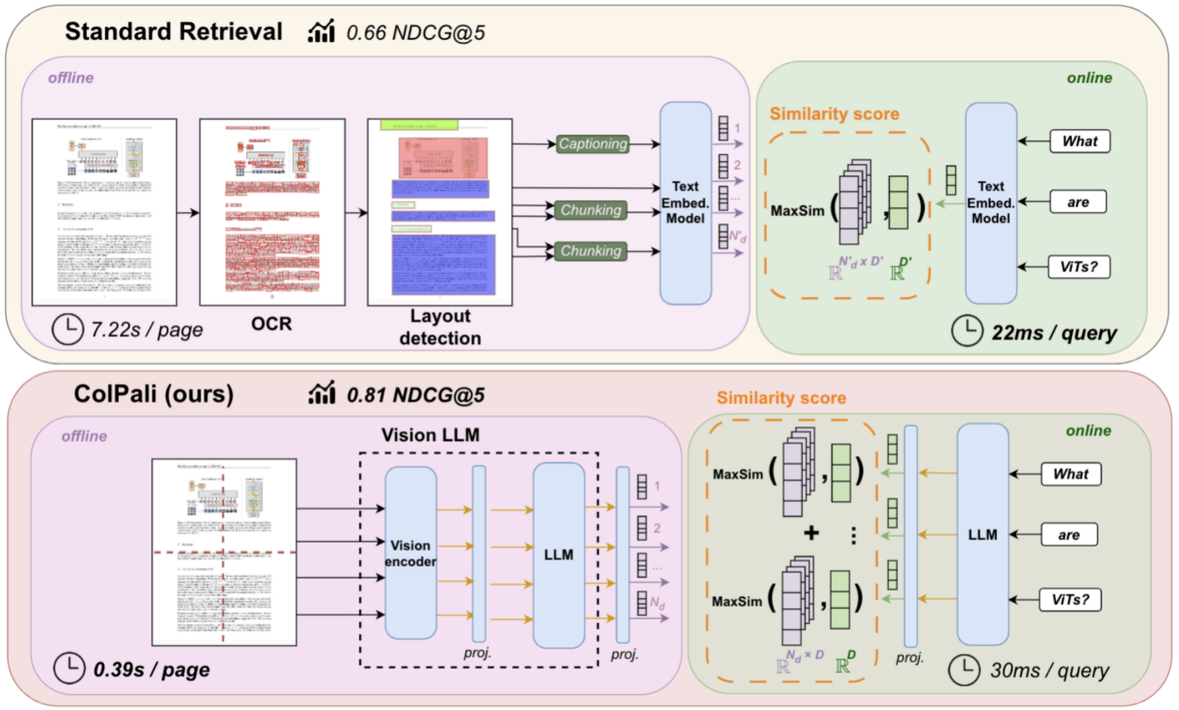
- Evaluation on BEIR / NFCorpus using ColBERTv2 as the multi-vector baseline.
- MUVERA configuration: $k_{sim} = 5$, $dim_{proj} = 16$, $r\_reps = 20$
- The results show that MUVERA-only search trades some accuracy for speed, achieving about 70% of the full multi-vector performance.
- Multistage pipeline recovers nearly all the original performance while maintaining the efficiency benefits for the initial search phase.
- MUVERA-only search is approximately 8x faster than full multi-vector search, while the hybrid approach with reranking still achieves about 7x speed improvement while maintaining nearly identical search quality.

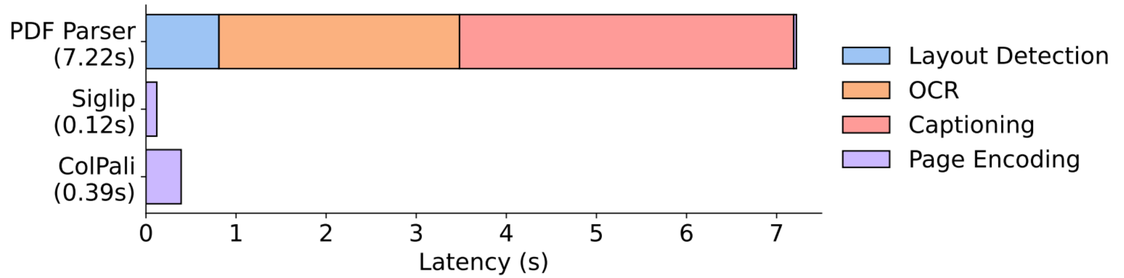
Approach	NDCG@1	NDCG@5	NDCG@10	Average search time
Full multi-vector (ColBERT)	0.478	0.387	0.347	1.27
MUVERA-only	0.319	0.267	0.242	0.15
MUVERA + reranking	0.475	0.383	0.343	0.18

MUVERA: Making multi-vectors more performant

Search your documents

- Traditional document retrieval in RAG/search systems relies on text extraction (OCR, layout detection) → text embedding pipelines
- These pipelines can be slow and complex, involving OCR, layout reconstruction, chunking
- Visually rich documents (PDFs with tables, figures, layouts) often lose important cues when converted to plain text
- ColPali proposes a vision-centred approach: embed the document page as an image, capturing text, layout, and visuals in a single model

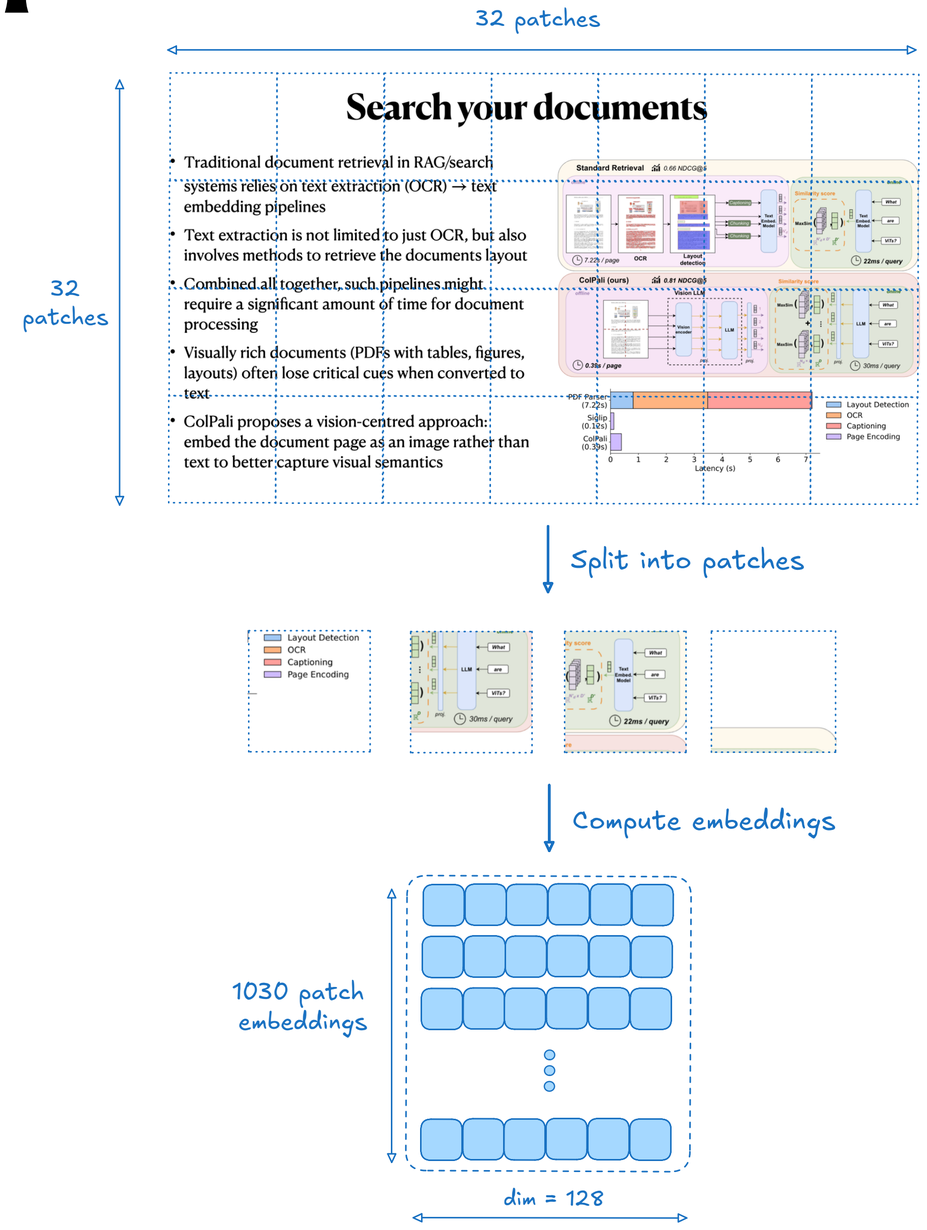




ColPali: Efficient Document Retrieval With Visual Language Models

ColPali

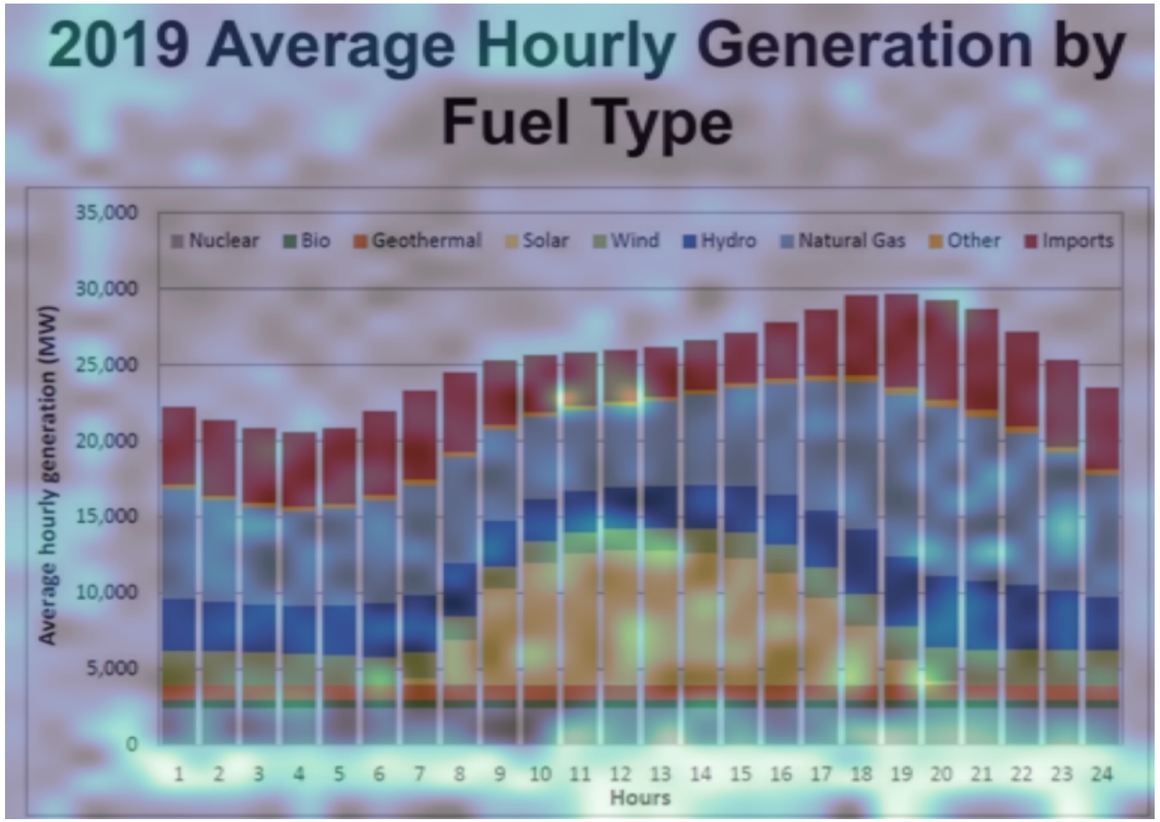
- ColPali uses a VLM (e.g., PaliGemma-3B) to generate multi-vector embeddings from document page images
- Each page is split into 32x32=1024 image patches; the model also uses a small number of prefix tokens (e.g., *<bos>Describe the image.)* for instruction/context.
- The VLM computes embeddings for all patch tokens and prefix tokens, producing ~1030 vectors per page
- Inspired by ColBERT, ColPali uses late interaction: each query vector matches its most similar document vector via MaxSim, and these scores are summed
- This setup replaces OCR + layout parsing + text embedding with a single image → embedding step in the retrieval pipeline



ColPali

Benefits & Performance

- ColPali outperforms text-centric retrievers on benchmarks like ViDoRe (visual document retrieval)
- Document preprocessing is faster than OCR & others
- It improves recall on documents where layout and visual structure matter
- Offers good interpretability via heatmaps
- However, generated multi-vectors have a large memory footprint and expensive to index
- As with ColBERT, building ANN indexes over thousands of vectors per document can be slow, so it's natural to use a multi-stage pipeline where a cheaper model prefetches candidates



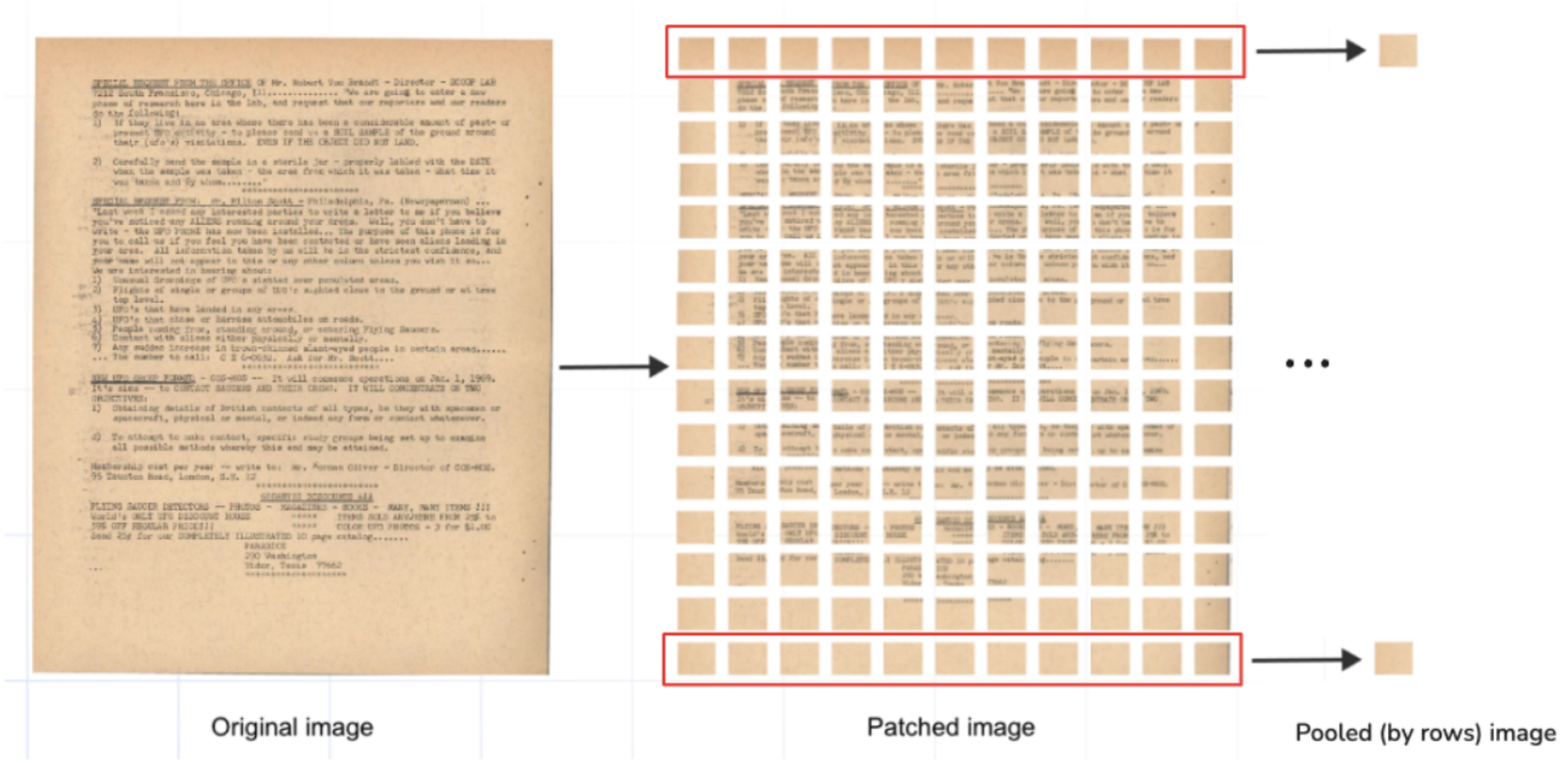
Query: "Which hour of the day had the highest overall electricity generation in 2019?"

ColPali: Efficient Document Retrieval With Visual Language Models

ColPali

Multistage pipeline

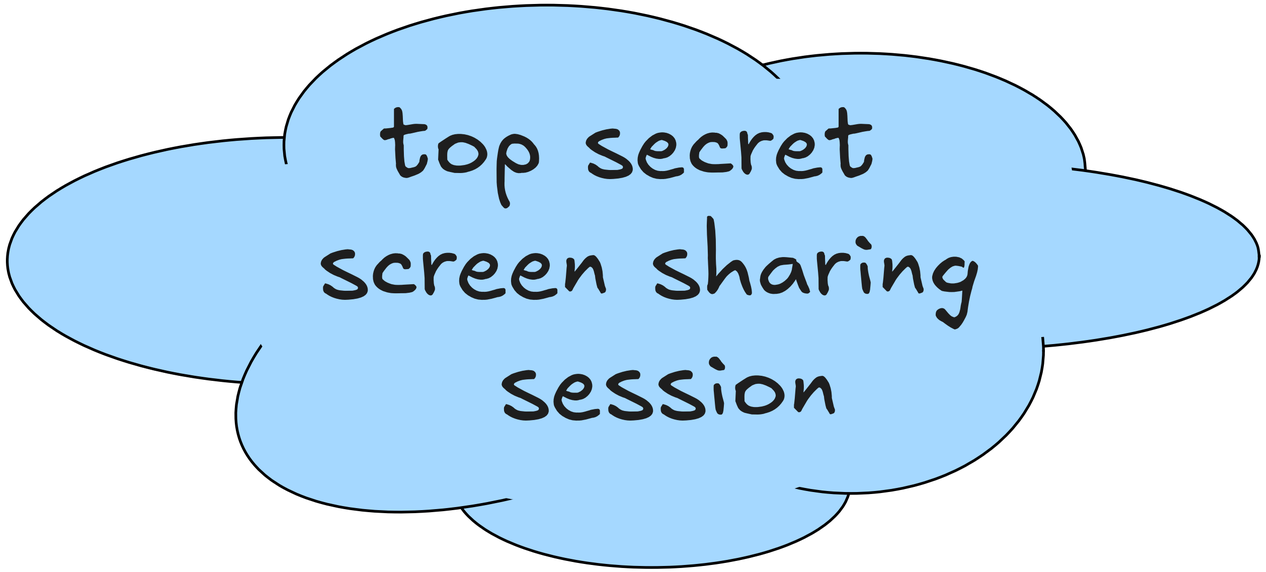
- As it was said, ColPali produces 1030 vectors per page
- Building an ANN index like HNSW on these vectors is expensive: with *ef_construct=100*, the total number of vector comparisons would be *1030*1030*100 =106 090 000* per page
- There are few alternative vision-based models that could serve as a cheaper first-stage retriever for this exact task, so a natural direction is to compress ColPali's own representations
- A practical approach is to pool ColPali embeddings by rows or by columns of the original document.
- Mean pooling reduces the number of vectors by a factor of 32, while maintaining high recall



PDF retrieval at scale

Demo in Qdrant

- We didn’t have enough time for the demo last time, let’s do it now



Questions?