ICEIS 2023 Indexing High-Dimensional Vector Streams

João Pedro V. Pinheiro[1], Lucas Ribeiro Borges[1], Bruno Francisco Martins da Silva[1], Luiz André P. Paes Leme[2], and Marco Antonio Casanova[1]

[1]Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro RJ, Brazil
[2]Universidade Federal Fluminense, Niterói RJ, Brazil

[1]{jpinheiro,lborges,bsilva,casanova}@inf.puc-rio.br
[2]lapaesleme@ic.uff.br

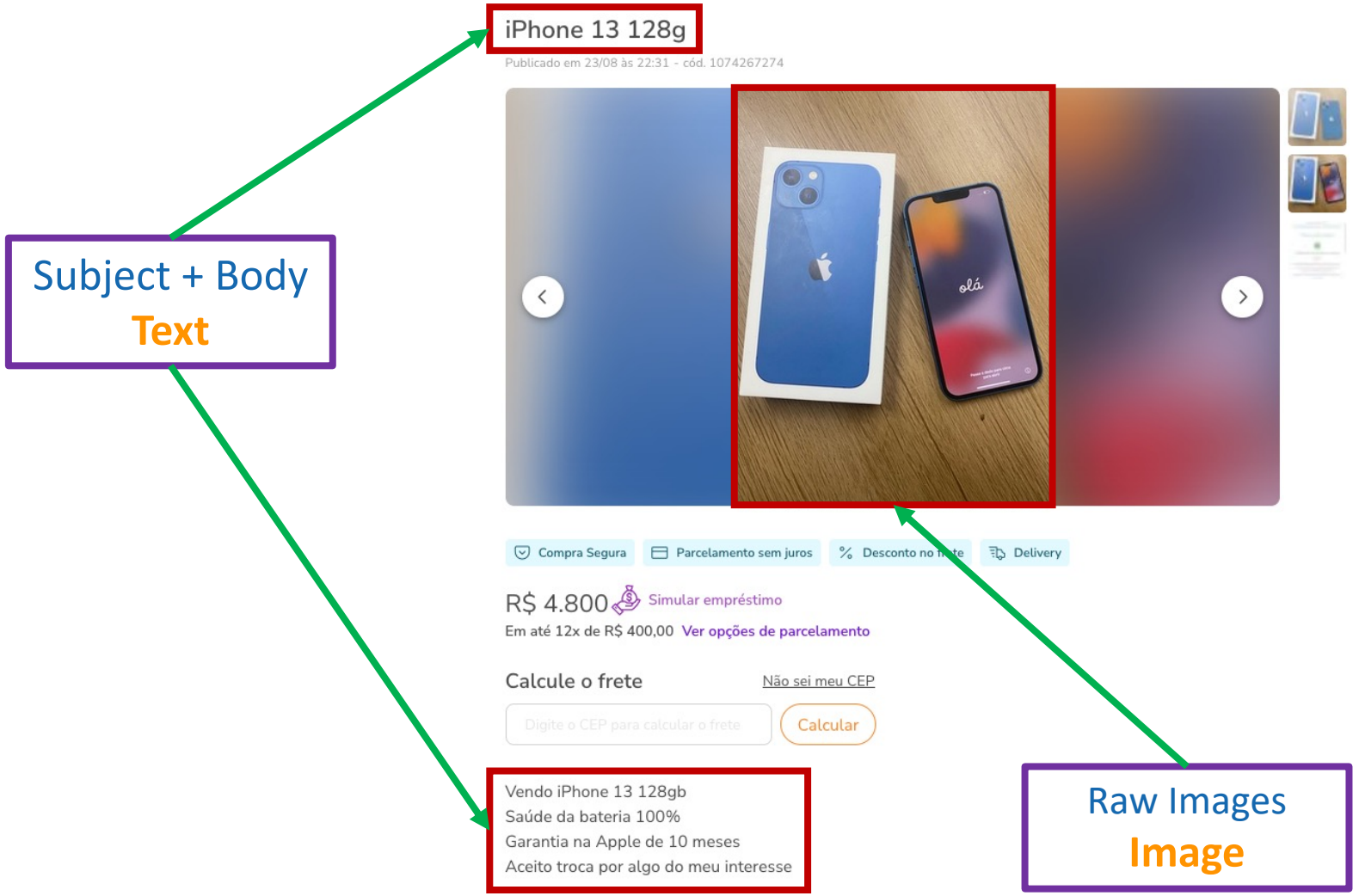
Introduction

Motivation

- The motivation for this paper lies in the challenge of creating a *classified ad retrieval tool* that receives a classified ad and returns a ranked list of similar ads
  - Similarity, in this case, would be computed with respect to the textual description, and the set of images of the classified ads
    - textual description: subject + body
    - set of raw images
  - The paper considers the scenario where new ads are *continuously included* in the platform, creating a *stream* of classified ads

Introduction

Motivation



Introduction
Challenges

- The construction of a classified ad retrieval tool, in this scenario, must face two major difficulties:
  - First, the tool would have to combine text and content-based image retrieval
    - albeit there are several well-tested text retrieval tools, retrieving images by content similarity is still challenging
    - the challenge then becomes how to efficiently search a (large set) of high-dimensional vectors or, more precisely, how to implement approximated nearest neighbor search over high-dimensional vectors
  - Second, the set of classified ads is dynamic
    - in the sense that sellers continuously create new ads, and ads may be short-lived
      - short-lived reasons: product sold, or the seller withdraw the ad, or the ad became obsolete
    - we model this scenario as a classified ad stream, where the retrieval process occurs over the stream, perhaps limited to some point in the past, as a *sliding time window*

Introduction

Problem definition

- This paper addresses the *vector stream similarity search problem*, defined as:
  - "Given a (high-dimensional) vector *q* and a time interval *T*, find a ranked list of vectors, retrieved from a vector stream, that are similar to *q* and that were received in the time interval *T*."

Staged Vector Stream Similarity Search
Definition

- The family of staged vector stream similarity search methods, or briefly SVS methods, refers to similarity search methods for vector streams with the following characteristics:
  - An SVS method uses a main memory cache C to store the vectors as they are received from the vector stream
  - When C becomes full, or a timeout occurs, the current stage terminates and the vectors in C are indexed and stored in secondary storage
  - The net result is a sequence of indexed sets of vectors, each set covering a specific time interval (sliding time window)
  - Hence, an SVS method does not depend on having the full set of vectors available beforehand, and it can cope with an unlimited number of vectors

Experiments

SVS methods analyzed

- The paper discusses experiments that assess the performance of implementations of two SVS methods:
  - IVFADC - "Inverted File with Asymmetric Distance Computation"[1], called staged IVFADC
  - HNSW – "Hierarchical Navigable Small World"[2] graphs, as implemented in Redis, and is called staged HNSW
- IVFADC and HNSW were chosen since they are well-known approximated vector similarity search methods

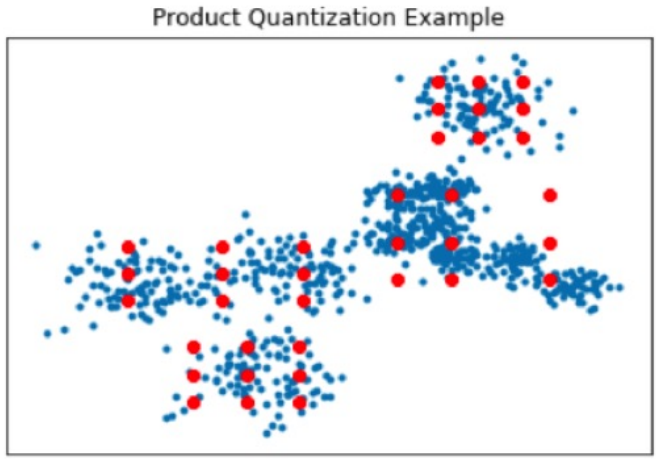
[1]Jégou, H., Douze, M., & Schmid, C. (2011). Product quantization for nearest neighbor search. IEEE Transactions on Pattern Analysis and Machine Intelligence, 33(1), 117–128. https://doi.org/10.1109/TPAMI.2010.57

[2]Malkov, Y. A. & Yashunin, D. A. (2020). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans. Pattern Anal. Mach. Intell., 42(4), 824–836. https://doi.org/10. 1109/TPAMI.2018.2889473

Experiments

Staged Product Quantization (*basic definition*)

- Product quantizer
  - a type of “vector quantizer” to accelerate approximate nearest neighbor search
  - a key element of the popular *Facebook AI Similarity Search* (*FAISS*) library
  - a k-NN search with a product quantizer performs an “exhaustive search” in some partitions of the dataset, but a product quantizer approximates and simplifies the distance calculations



Run product quantization on each coarse cell separately

Experiments

Staged Product Quantization (*IVFADC features*)

- Compact coding scheme provides an accurate approximation of the Euclidean distance
- Coding scheme is combined with an inverted file system to avoid exhaustive search, resulting in high efficiency
- Significantly outperforms the state of the art in terms of the trade-off between search quality and memory usage
- Scalability validated on a data set of two billion vectors

Experiments

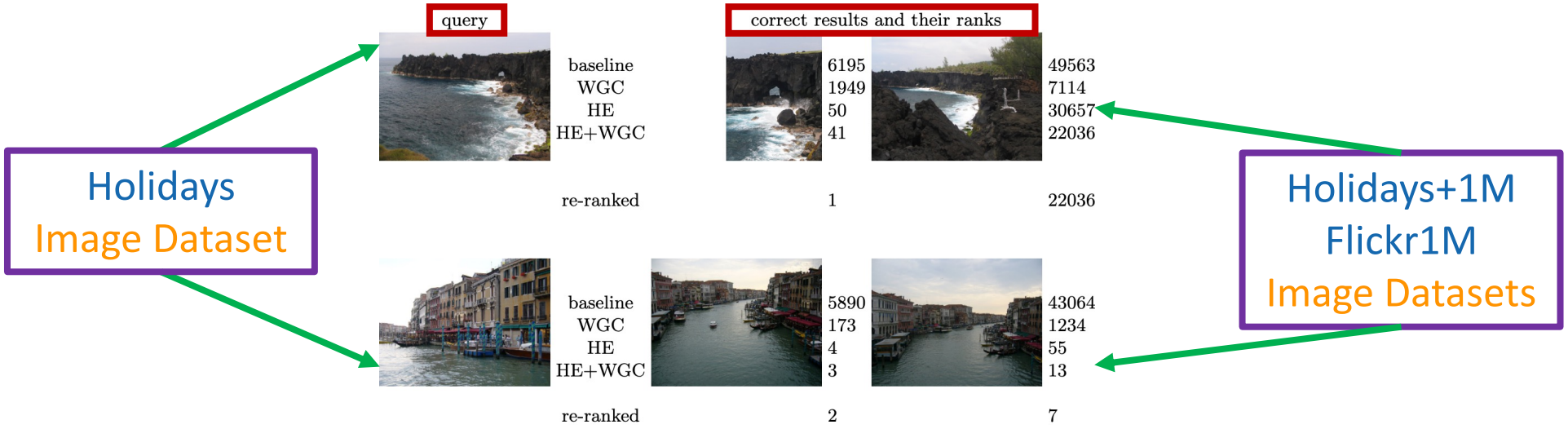
Staged Product Quantization (*methodology*)

- The experiments assess the performance of the IVFADC implementation of the staged ingestion of a stream of vectors with deferred indexing (IVFADC-SD)
- In more detail, the goals of the experiments are:
  - build cost: evaluate the cost of IVFADC-SD, for various cache sizes
  - query cost: compare the cost of IVFADC-SD with a baseline when processing query sets
  - search quality: compare the mean average *recall@R* of IVFADC-SD with that of a baseline when processing a set of queries

Experiments

Staged Product Quantization (*methodology*)

- The experiments use:
  - base dataset: a random partition of the 1 million INRIA Holidays images into 10 batches (B1, ..., B10)
  - query descriptors: from the INRIA Holidays images



Experiments

Staged Product Quantization (*methodology*)

- Since the original experiments - used as baseline - adopted Euclidean distance as distance metric, and mean average *recall@R* as quality metric, we also adopted in our work
- In all experiments, the codebooks and the vectors were stored in main memory
- The experiments used a PC with an Intel core i5-9600k CPU @ 3.7GHz processor and 16 GB RAM (12GB for the VM), running Ubuntu 20.04 on WSL2 VM and python 3.10

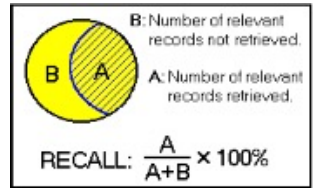
Experiments

Staged Product Quantization (*baseline metrics comparison*)

- The results were:
  - IVFADC-SD and the baseline IVFADC had equivalent total build cost, both in terms of training the codebooks and indexing the data
  - IVFADC-SD had a significant increase of around 40% in query cost when querying across all 10 batches
  - Search quality, measured by mean average *recall@R*, saw a slight reduction for lower values of R, but was otherwise similar to the baseline

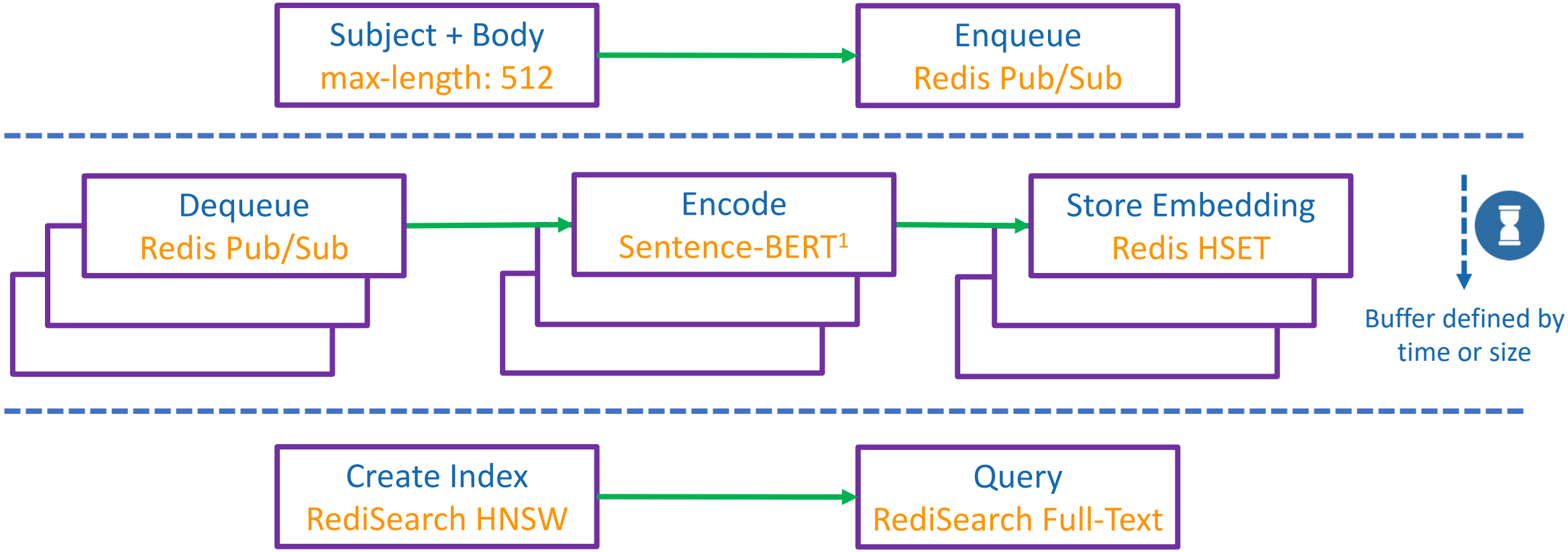
Table 4: Recall values of the staged product quantization experiments.

	recall@1	recall@5	recall@10	recall@20	recall@50	recall@100
Baseline	0.269	0.534	0.646	0.743	0.826	0.859
IVFADC-SD	0.251	0.504	0.618	0.725	0.823	0.865



Experiments

Staged HNSW (*indexing and searching pipelines*)



[1]Reimers, Nils and Gurevych, Iryna (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing. http://arxiv.org/abs/1908.10084

Experiments

Staged HNSW (*dataset from classified ads company*)

- Overview of daily ads volume
  - 444k ads inserted daily on average (~5/sec)
  - Telephony and Cellphones ads were considered only
    - may expand to other verticals in future work
    - attributes:
      - caption [subject + body]
      - raw images also in future work

category_id_fk	category_name	approved_ads	%
2	Apartamentos	1210056	12,86
86	Terrenos, sítios e fazendas	596255	6,34
3	Casas	593646	6,31
30	Móveis	558518	5,94
5	Celulares e telefonia	554264	5,89
46	Carros, vans e utilitários	480869	5,11
55	Computadores e acessórios	391480	4,16
68	Roupas e calçados	375598	3,99
89	Eletrodomésticos	345318	3,67

day	all_ads	approved_ads	approval_tax	weekday
22	484537	433759	0,9	Monday
21	306672	283170	0,92	Sunday
20	621361	596956	0,96	Saturday
19	422226	390366	0,92	Friday
18	459418	422443	0,92	Monday
17	506167	468939	0,93	Sunday
16	521264	479947	0,92	Saturday
15	474828	438588	0,92	Friday
14	250571	232412	0,93	Monday
13	368705	346709	0,94	Sunday
12	449885	420711	0,94	Saturday
11	482976	449079	0,93	Friday
10	490995	456296	0,93	Monday
9	524239	488014	0,93	Sunday
8	501021	465998	0,93	Saturday
7	290010	268749	0,93	Friday
6	363546	341405	0,94	Monday
5	462015	433573	0,94	Sunday
4	522394	489245	0,94	Saturday
3	519268	484896	0,93	Friday
2	575599	539436	0,94	Monday
1	513927	478495	0,93	Sunday
	Avg.:	443833,50	0,93	
	Avg./Hour	18493,06		
	Avg./Min	308,22		
	Avg./Sec	5,14		

Experiments

Staged HNSW (*methodology*)

- Data extraction occurred between 2022-Jun and 2022-Jul
- The extracted data was split into batches and used in experiments
  - batch sizes: 50k, 100k, 250k, 500k, 750k, and 1MM
  - vectors obtained from encoding the ad texts
    - all with the same dimension: 768 (classic BERT output)
  - HNSW was simulated in Redis
  - FLAT algorithm was used as baseline also in Redis
    - full sequential scan of the embeddings (exhaustive search)
- Redis was run on a PC server with OS GNU/Linux Ubuntu 16.04.6 LTS, a quad-core processor Intel(R) Core(TM) i7-5820K CPU @ 3.30GHz, with 64 GB of RAM and 1TB of SSD

Experiments

Staged HNSW (*indexing and query times*)

Table 5: Indexing times (in ms) for various dataset and batch sizes (text-only).

Dataset size	Batch size	Batch 0	Batch 1	Batch 2	Batch 3	Batch 4	All batches	HNSW
50,000	10,000	7,374	7,883	7,388	7,333	***7,226***	37,204	97,376
100,000	20,000	16,231	15,348	13,251	14,859	13,912	73,601	201,870
250,000	50,000	68,378	63,417	67,307	63,645	103,408	366,155	536,274
500,000	100,000	188,794	252,109	155,201	159,594	105,520	861,218	1,242,132
1,000,000	200,000	244,318	234,918	233,393	232,553	234,731	1,179,913	2,128,172

Table 6: Query processing times (in ms; dataset size = 1,000,000).

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Avg
FLAT (k=10)	3.4132	0.2694	0.2859	***0.2629***	0.2718	0.2855	0.2800	0.2687	0.3117	0.2714	0.2806
HNSW (k=10)	0.1241	0.0523	0.0935	***0.0463***	0.0697	0.0871	0.0793	0.0623	0.1401	0.0684	0.0796
HNSW batch 0 (k=4)	0.0831	0.0444	0.0676	***0.0394***	0.0534	0.0664	0.0600	0.0487	0.0925	0.0526	0.0595
HNSW batch 1 (k=4)	0.0689	0.0407	0.0577	***0.0358***	0.0462	0.0571	0.0519	0.0421	0.0776	0.0457	0.0513
HNSW batch 2 (k=4)	0.0660	0.0385	0.0533	***0.0335***	0.0412	0.0522	0.0470	0.0382	0.0764	0.0415	0.0472
HNSW batch 3 (k=4)	0.0660	0.0368	0.0521	***0.0316***	0.0382	0.0506	0.0457	0.0356	0.0765	0.0385	0.0454
HNSW batch 4 (k=4)	0.0660	0.0358	0.0519	***0.0311***	0.0380	0.0528	0.0454	0.0341	0.0765	0.0373	0.0452

Indexing
FLAT(n=1M) = 12,718ms
92.8x faster than *All batches* (~19min)
167.3x faster than *HNSW full* (~35min)

Indexing and Query
Euclidean distance metric
k=10 (max allowed on Redis)

Query
HNSW vs FLAT
HNSW full 3.5x faster than FLAT
HNSW batches 4.7-6.2x faster than FLAT

Experiments

Staged HNSW (*baseline metrics comparison*)

Table 7: Precision values of the query experiments.

		Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Avg
HNSW	**precision@1**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	precision@5	0.20	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.92
	precision@10	0.50	0.90	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.94
HNSW batches	**precision@1**	1.00	0.00	1.00	0.00	0.00	1.00	0.00	1.00	0.00	1.00	0.50
	precision@5	0.40	0.40	0.60	0.20	0.40	0.40	0.60	0.40	0.60	1.00	0.50
	precision@10	0.50	0.50	0.60	0.40	0.30	0.40	0.60	0.30	0.50	0.80	0.49

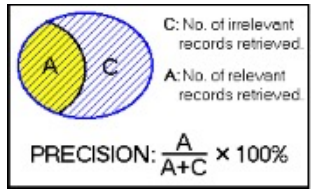
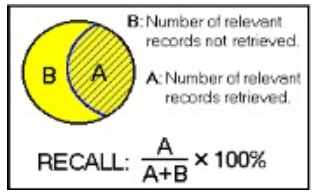


Table 8: Recall values of the query experiments.

		Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Avg
HNSW	**recall@1**	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10	0.10
	recall@5	0.10	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.46
	recall@10	0.50	0.90	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.94
HNSW batches	**recall@1**	0.10	0.00	0.10	0.00	0.00	0.10	0.00	0.10	0.00	0.10	0.05
	recall@5	0.20	0.20	0.30	0.10	0.20	0.20	0.30	0.20	0.30	0.50	0.25
	recall@10	0.50	0.50	0.60	0.40	0.30	0.40	0.60	0.30	0.50	0.80	0.49



Q1 - 1045201614		
FLAT	HNSW	HNSW buckets
1045201614	1045201614	1045201614
1042000475	1039916370	1040329529
1043822312	1048630808	1039916370
1040329529	1045704713	1049301862
1054653176	1047586828	1039700012
1041072980	1045962608	1045584577
1039916370	1046041394	1047586828
1048630808	1045584577	1045283863
1045704713	1046230724	1044614193
1047586828	1044614193	1048630808

Q2 - 1045225950		
FLAT	HNSW	HNSW buckets
1045225950	1045225950	1045074523
1045074523	1045074523	1041571347
1041571347	1041571347	1046116836
1044805645	1044805645	1044837878
1048658078	1048658078	1048104184
1040591965	1040591965	1048634082
1046116836	1044837878	1048600254
1044837878	1051937272	1052883862
1051937272	1048104184	1039734035
1048104184	1048175581	1053457940

Q3 - 1045273915		
FLAT	HNSW	HNSW buckets
1045273915	1045273915	1045273915
1044474681	1044474681	1039761010
1039761010	1039761010	1047115949
1047115949	1047115949	1046005369
1045398557	1045398557	1056278724
1054337360	1054337360	1045598813
1046005369	1046005369	1044629730
1056278724	1056278724	1047594933
1045598813	1045598813	1041538100
1040271801	1040271801	1042793089

A Classified Ad Retrieval Tool

Proof-of-Concept

- As an example of the retrieval output, suppose that the user wants to find the top 3 ads most similar to the following ad
  - “Sell Motorola E7. Sell Motorola E7, three months of use, with invoice, and cover and glass protection cover”
- The tool will return
  - “Motorola e7 in good conditions. Sell Motorola e7 with 4 months of use invoice and everything”
  - “Motorola E7 almost new in the box. Sell Motorola E7 in the box for 700.00 4 months of use”
  - “Motorola E7 only today. Sell Motorola E7 for 500 reais together with. Original charger Original earphone invoice and box. It will be 8 months old. Reason *****. ZAP.********”

Conclusions

Contributions and future work

- Contributions of this paper:
  - a family of algorithms, called staged vector stream similarity search – SVS, to dynamically index a stream of high-dimensional vectors and facilitate similarity search
  - a stream of vectors that become obsolete over time requires an approach different from static vector indexing methods or updating such data structures
  - two different experiments suggesting that the SVS implementations do not incur significant overhead and achieve reasonable search quality

Conclusions

Contributions and future work

- Next steps:
  - we plan to conduct further experiments with the proof-of-concept retrieval tool, using much larger datasets collected from the classified ad platform and larger sets of realistic queries
  - improve experiments over images comparing indexing process with different deep learning techniques available
    - single high-dimensional vector for multimodal inputs
    - different high-dimensional vector for each modal (image and text)

Thank you!

João Pedro V. Pinheiro[1], Lucas Ribeiro Borges[1], Bruno Francisco Martins da Silva[1], Luiz André P. Paes Leme[2], and Marco Antonio Casanova[1]

[1]Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro RJ, Brazil
[2]Universidade Federal Fluminense, Niterói RJ, Brazil

[1]{jpinheiro,lborges,bsilva,casanova}@inf.puc-rio.br
[2]lapaesleme@ic.uff.br