REAL-WORLD
DATASETS DISTRIBUTION

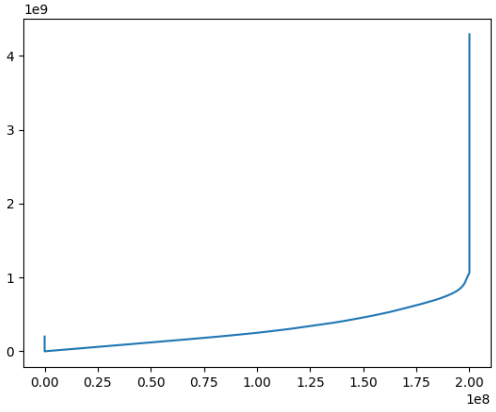


Figure 1: Amzn uint32

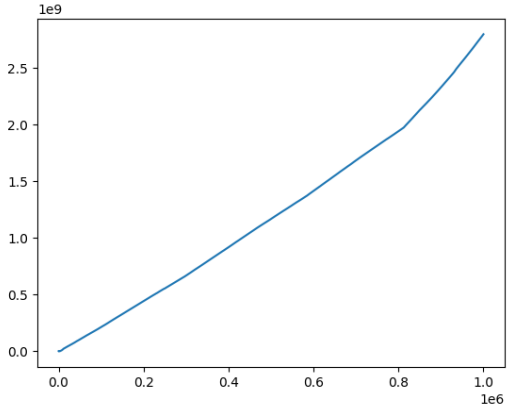


Figure 2: CompanyNet

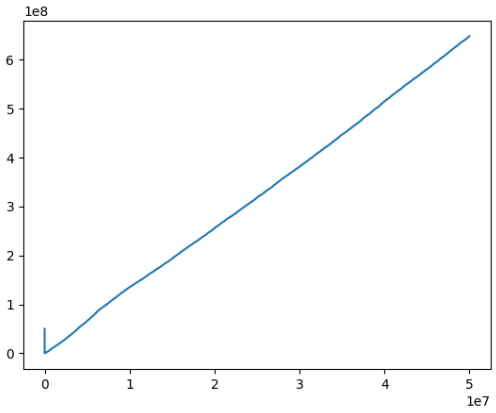


Figure 3: Friendster

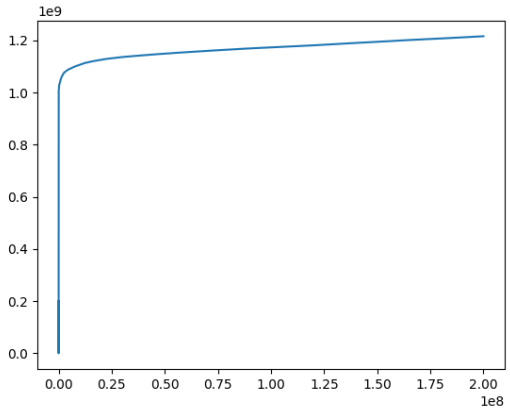


Figure 4: Wiki uint32

Real-world datasets - distribution. The x axis reports the indexes (positions) in the vector to store, while the y axis reports the corresponding values.

SYNTHETIC DATASETS
DISTRIBUTION

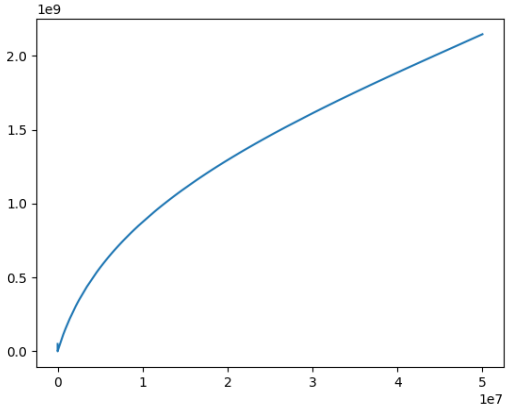


Figure 5: Normal

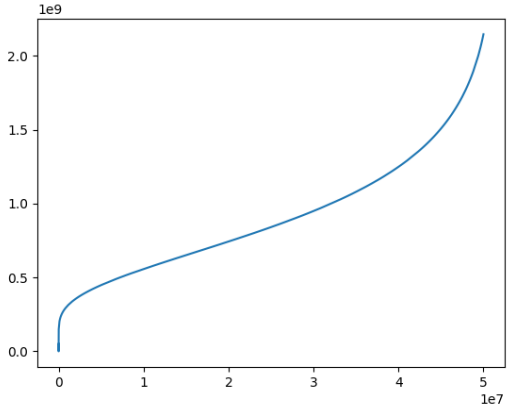


Figure 6: Lognormal

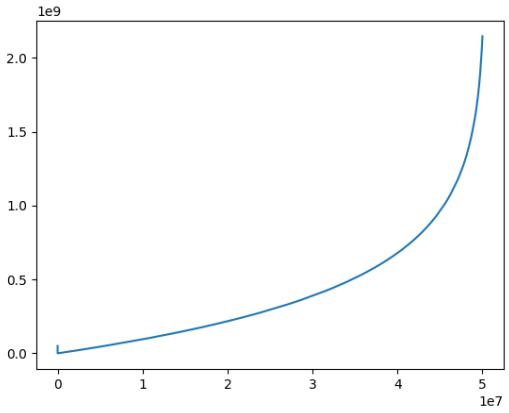


Figure 7: Exponential

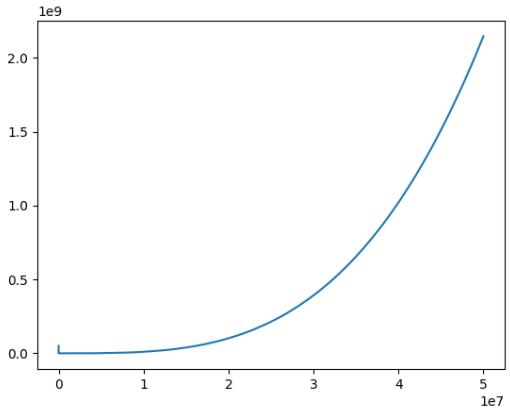


Figure 8: Zipf

Synthetic dataset - distribution. The x axis reports the indexes (positions) in the vector to store, while the y axis reports the corresponding values.

64-BIT DATASETS
DISTRIBUTION

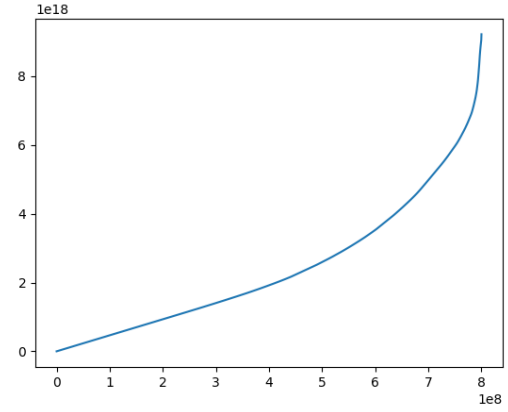


Figure 9: Amzn uint64

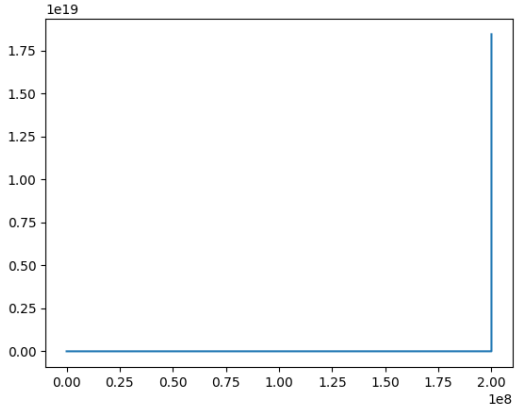


Figure 10: Facebook uint64

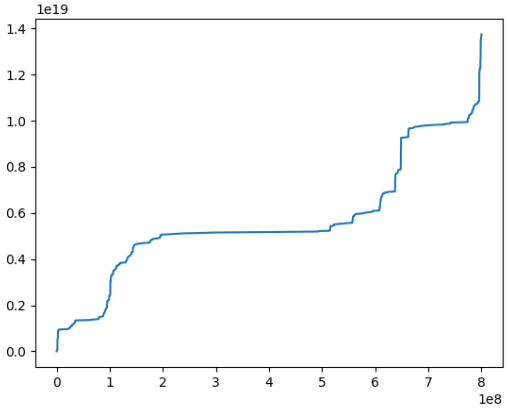


Figure 11: OSM Cellids uint64

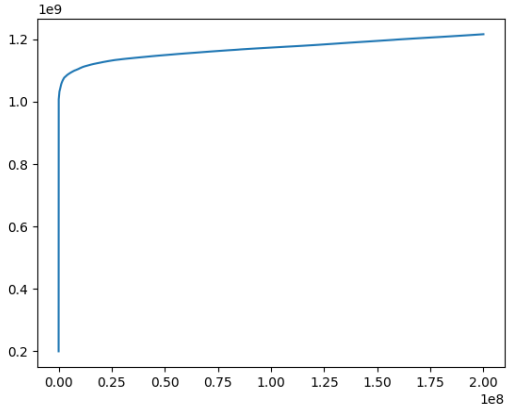
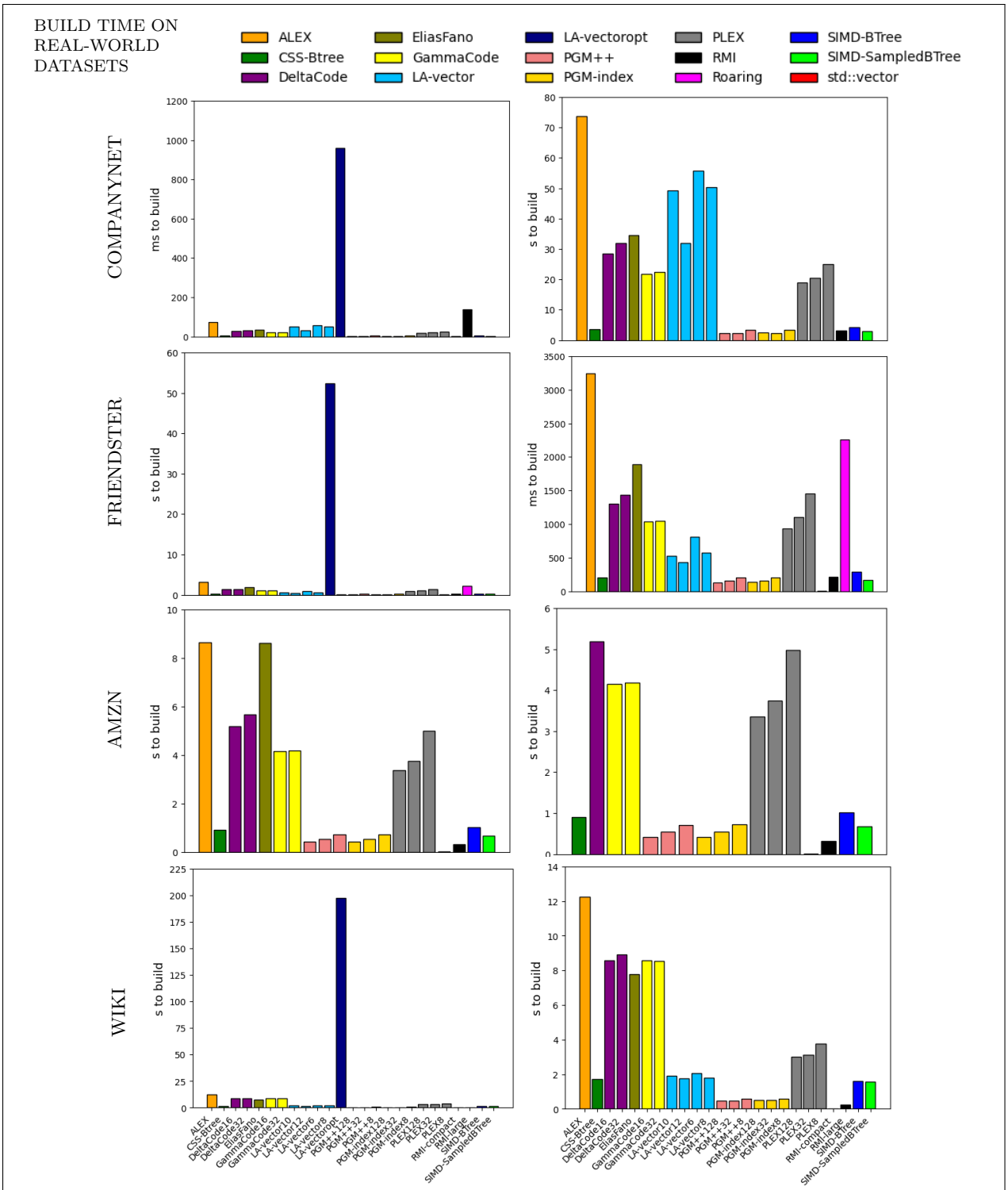


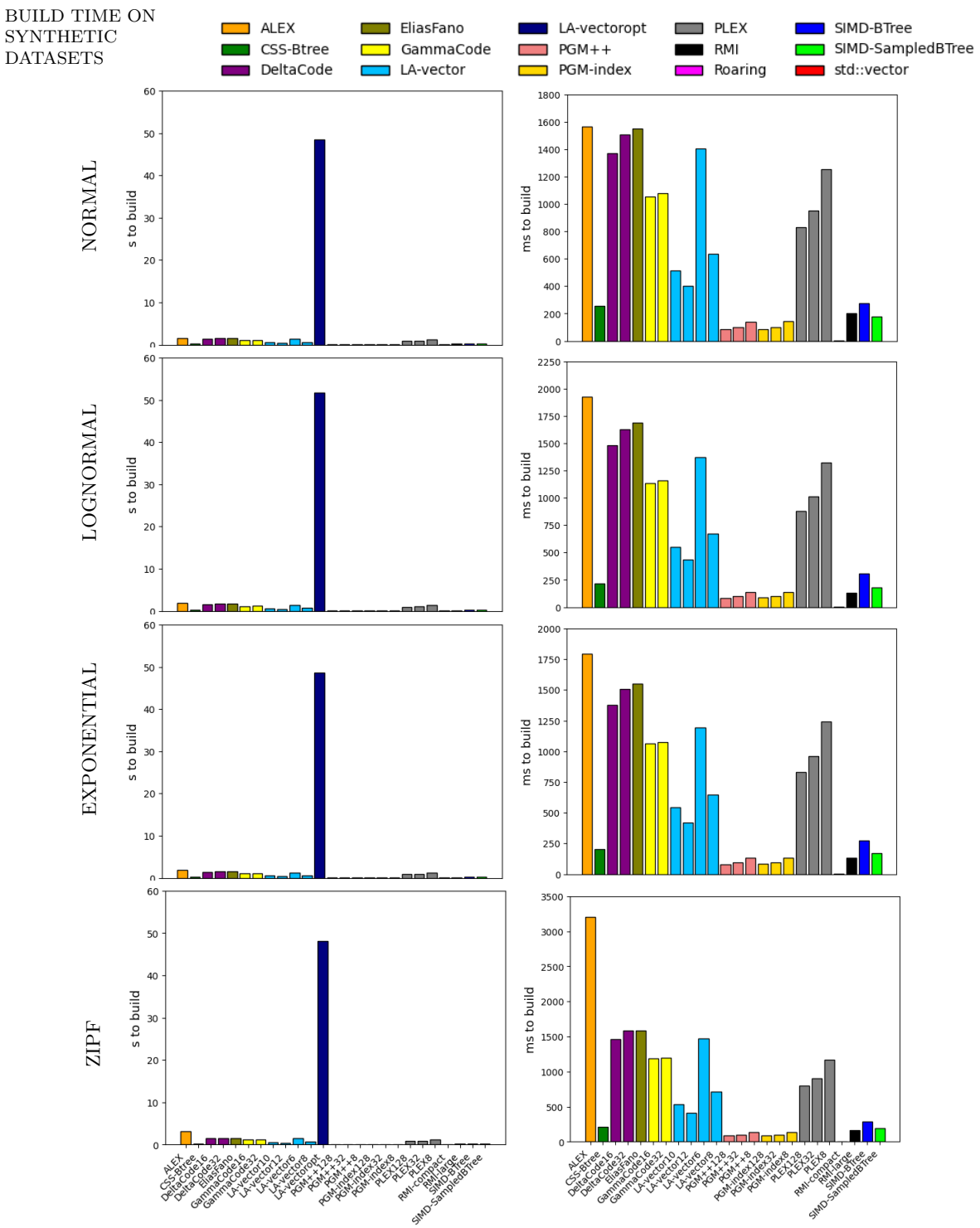
Figure 12: Wiki uint64

64-bits datasets - dataset distributions. The x axis reports the indexes (positions) in the vector to store, while the y axis reports the corresponding values.



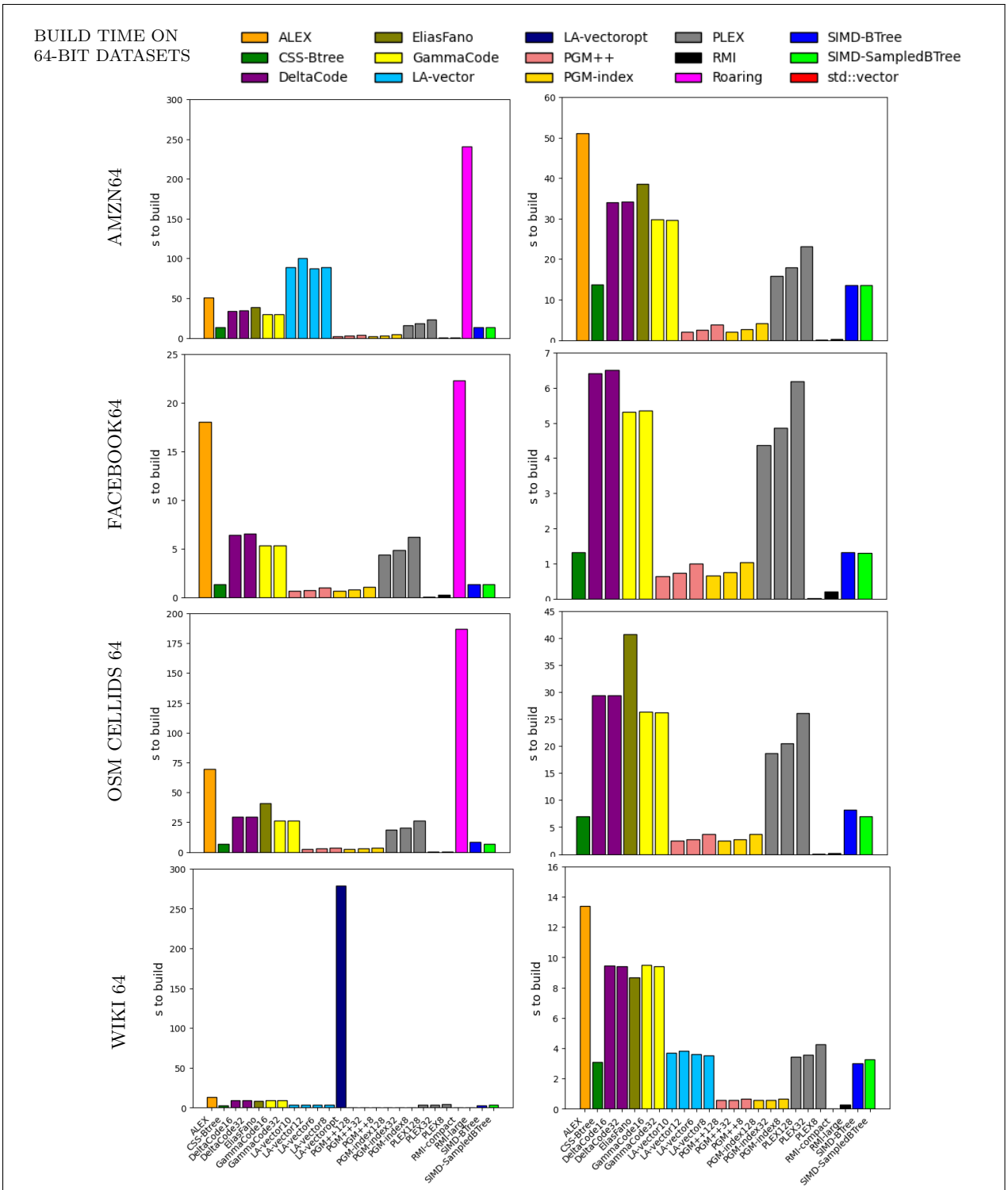
The plots above report the time required to build the data structures (indexes) on the real-world datasets. On the left, the dataset's name is shown; in the middle, the complete plot with all the times expressed in milliseconds; and on the right, the same plot but excluding the ones with excessive build times (those that are over double the average of all build times of all indexes).

Figure 13: Build time on real-world datasets



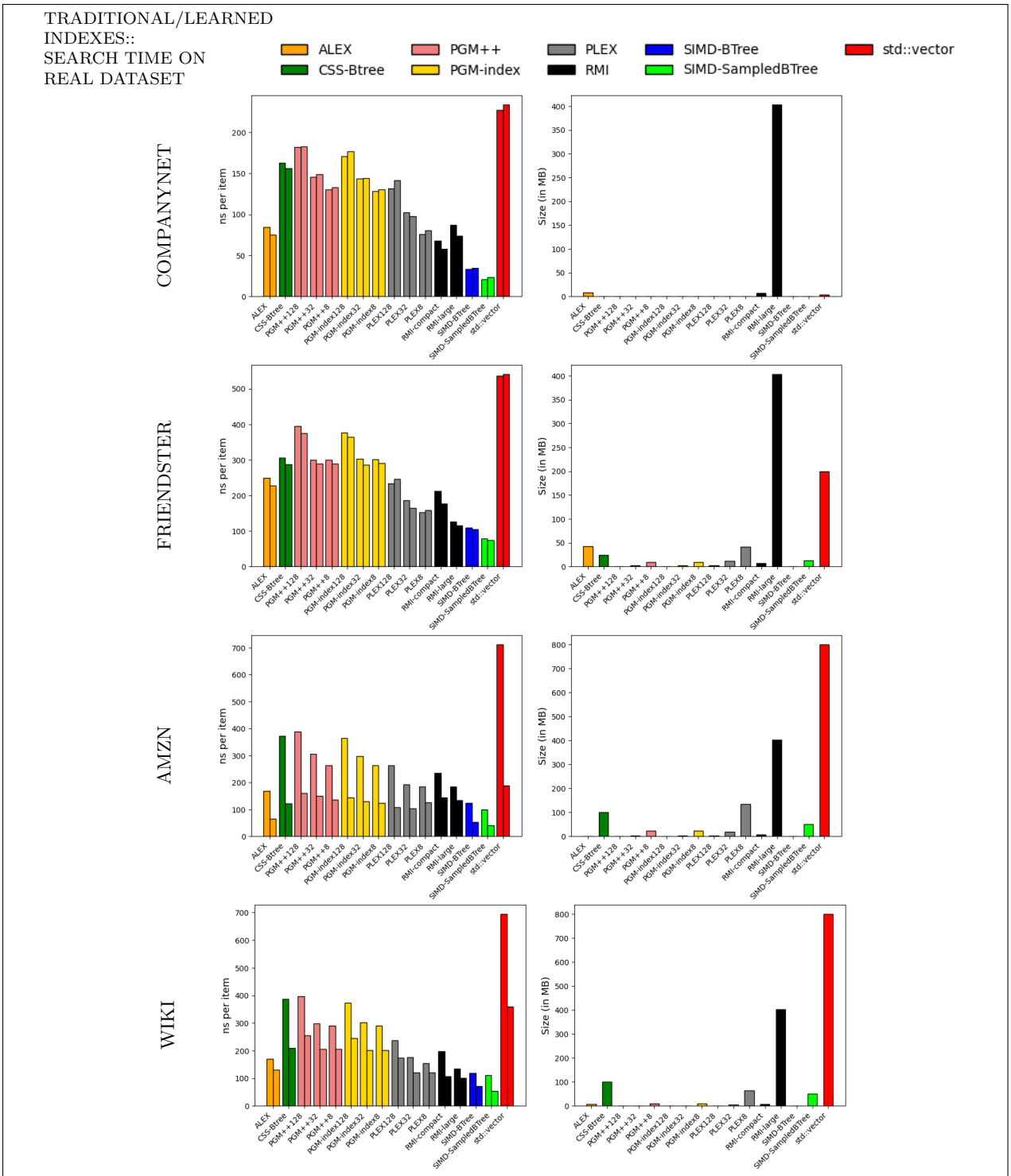
The plots above report the time required to build the data structures (indexes) on the synthetic datasets. On the left, the dataset's name is shown; in the middle, the complete plot with all the times expressed in milliseconds; and on the right, the same plot but excluding the ones with excessive build times (those that are over double the average of all build times of all indexes).

Figure 14: Build time on synthetic datasets



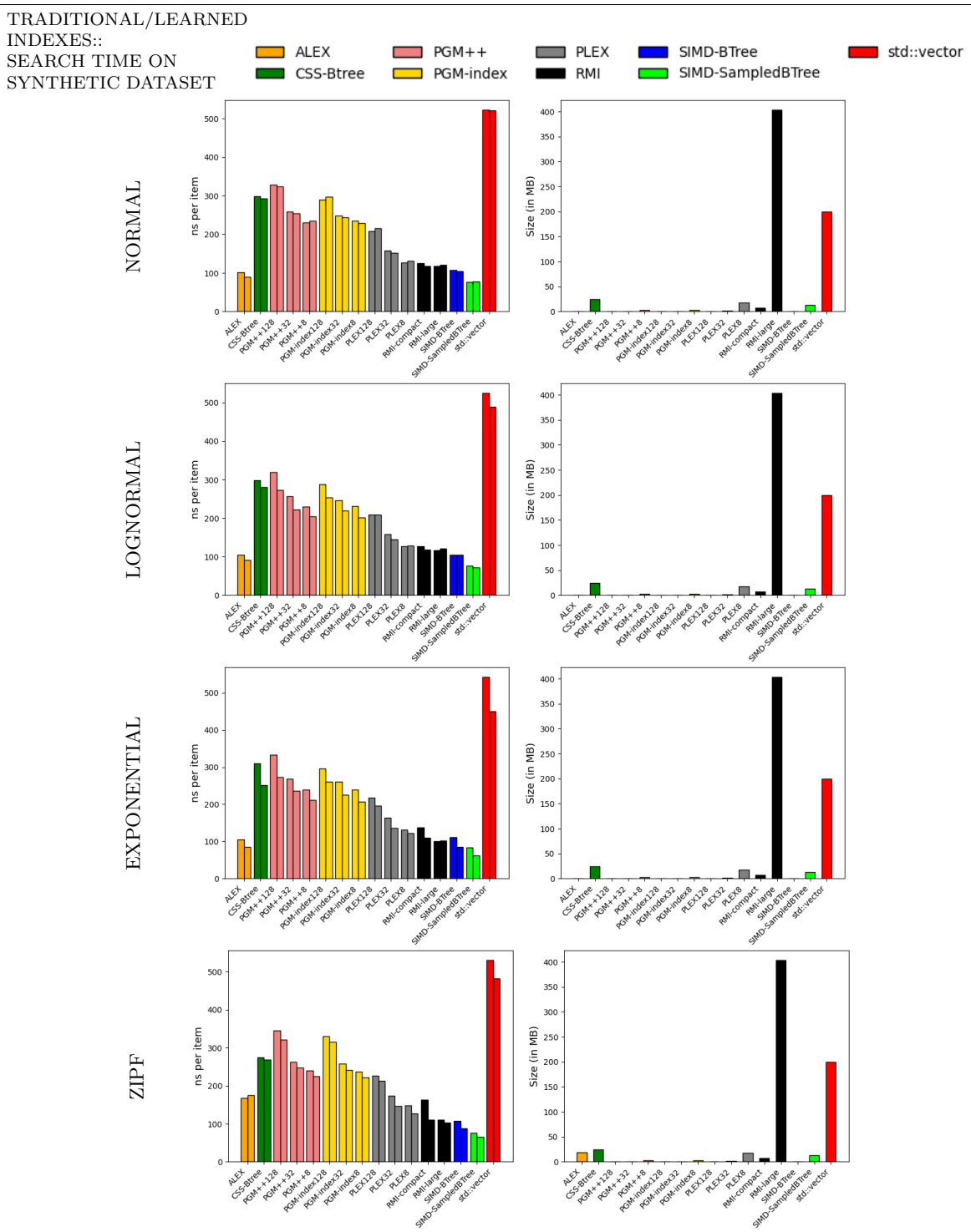
The plots above report the time required to build the data structures (indexes) on the 64-bit datasets. On the left, the dataset's name is shown; in the middle, the complete plot with all the times expressed in milliseconds; and on the right, the same plot but excluding the ones with excessive build times (those that are over double the average of all build times of all indexes).

Figure 15: Build time on 64-bit datasets



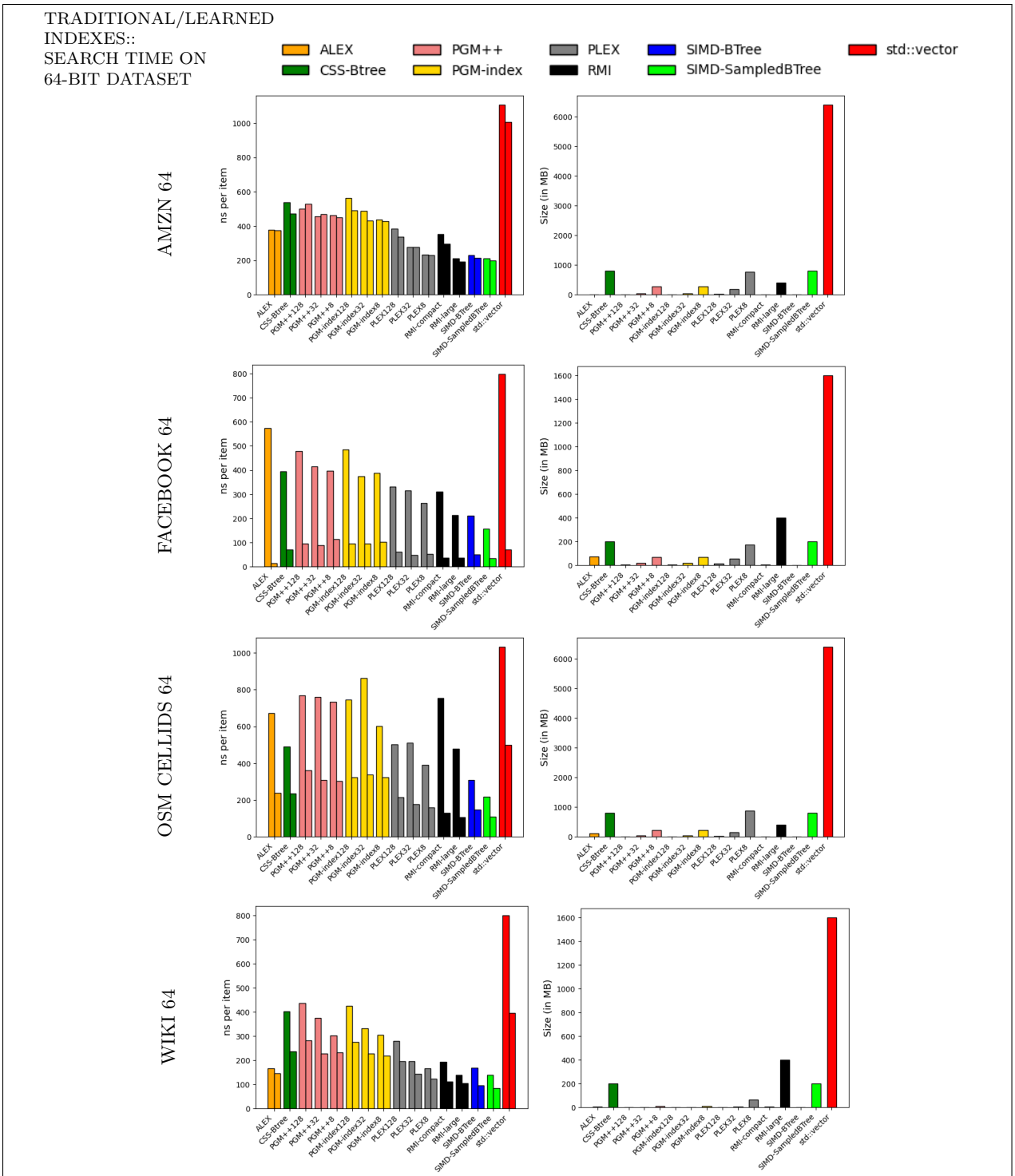
The performances of traditional and learned indexes on real-world datasets are represented in two plots. The leftmost plot shows the time of search (in ns) on each index; while the right plot shows the space occupied by each index in MBs (without considering the space occupied by the data to be stored). The left plot shows two bars for each data structure. The left/right one shows the average time to search for an existing/missing item in the collection.

Figure 16: Average time for pointwise queries on traditional and learned indexes, built on real-world datasets.



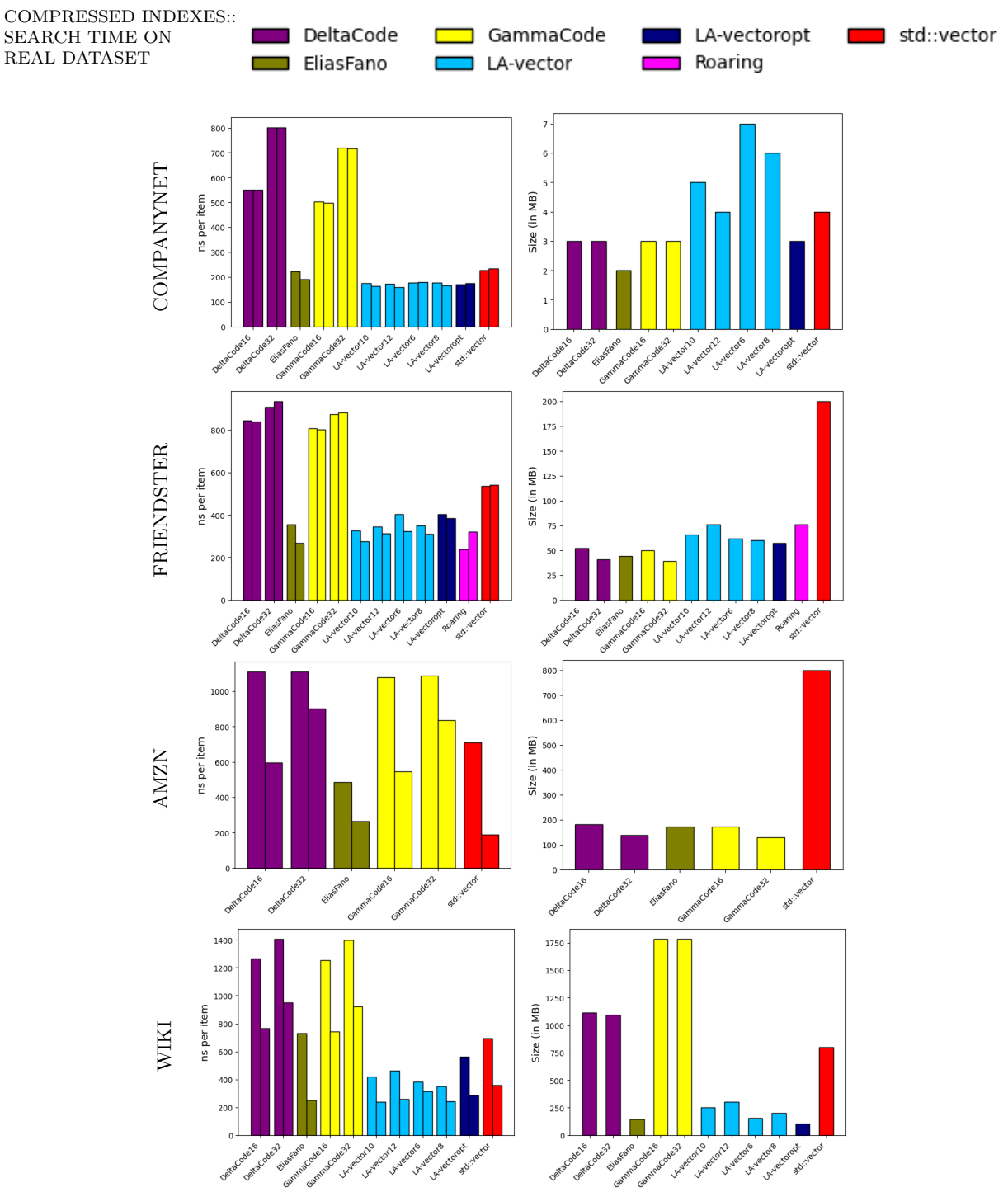
The performance of traditional and learned indexes on synthetic datasets are represented in two plots. The leftmost plot shows the time of search (in ns) on each index; while the right plot shows the space occupied by each index in MBs (without considering the space occupied by the data to be stored). The left plot shows two bars for each data structure. The left/right one shows the average time to search for an existing/missing item in the collection.

Figure 17: Average time for pointwise queries on traditional and learned indexes, built on synthetic datasets.



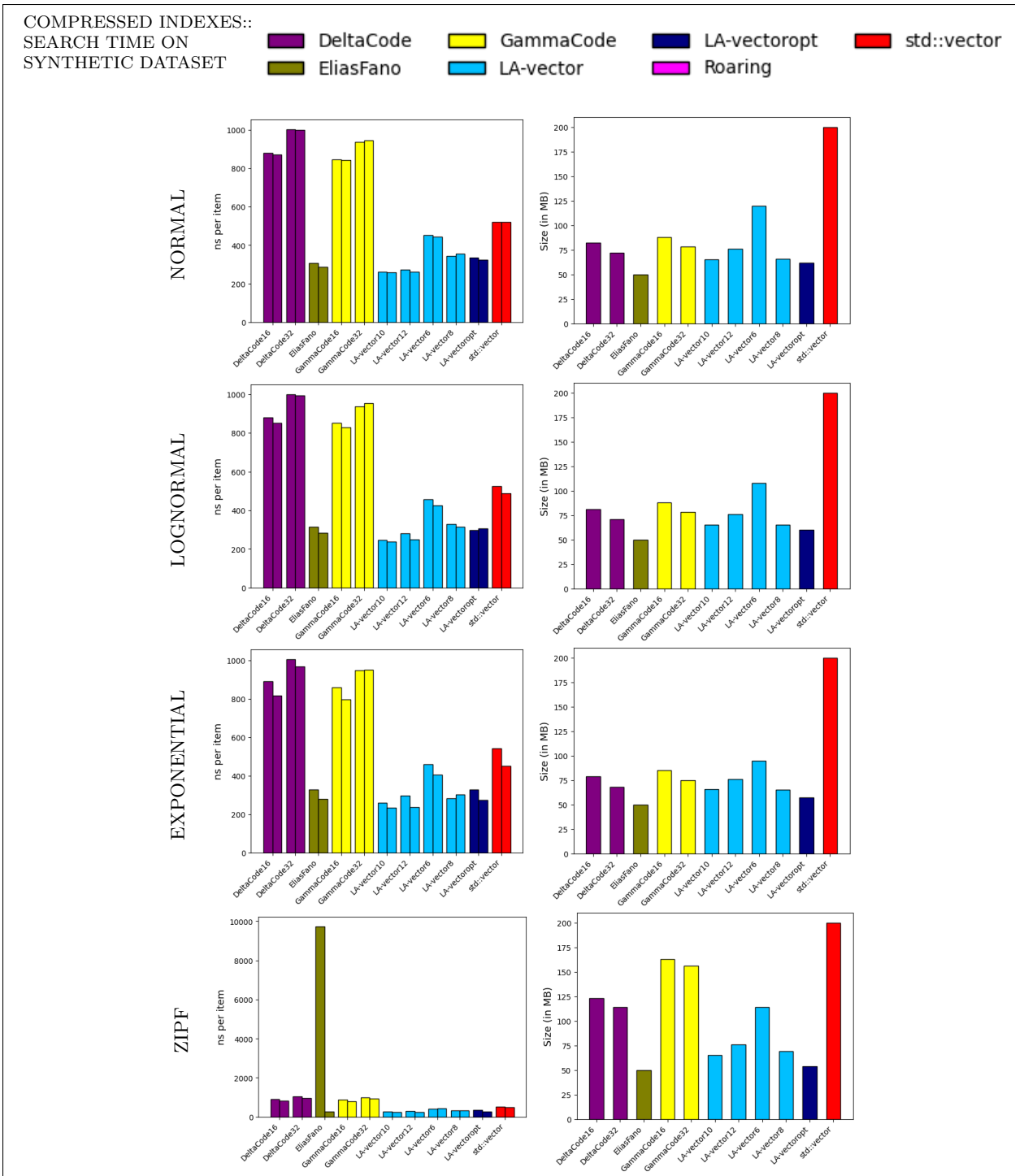
The performances of traditional and learned indexes on 64-bit datasets are represented in two plots. The leftmost plot shows the time of search (in ns) on each index; while the right plot shows the space occupied by each index in MBs (without considering the space occupied by the data to be stored). The left plot shows two bars for each data structure. The left/right one shows the average time to search for an existing/missing item in the collection.

Figure 18: Average time for pointwise queries on traditional and learned indexes, built on 64-bit datasets.



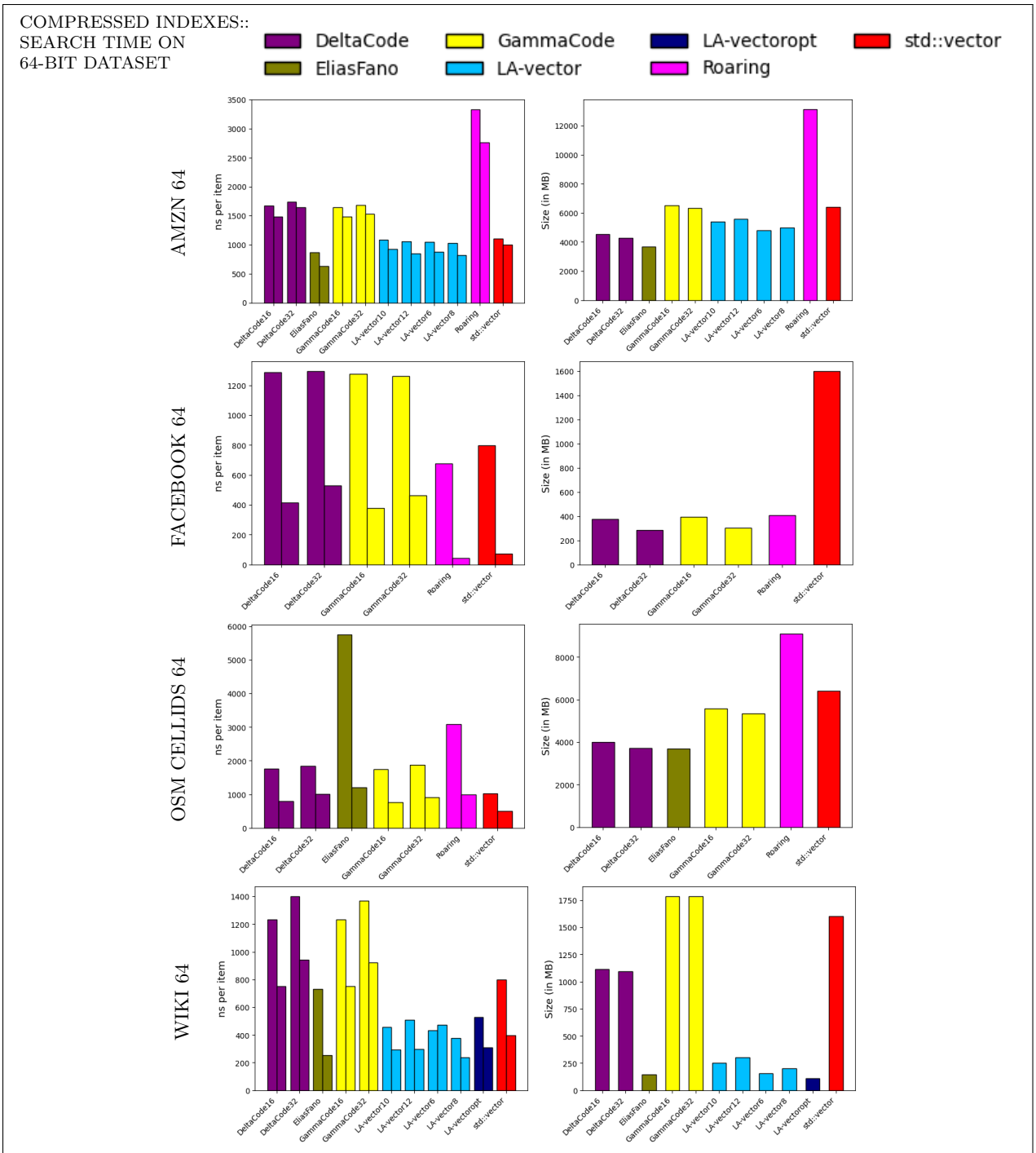
The performances of compressed indexes on real-world datasets are represented in two plots. The leftmost plot shows the time of search (in ns) on each index; while the right plot shows the space occupied by each index in MBs (without considering the space occupied by the data to be stored). The left plot shows two bars for each data structure. The left/right one shows the average time to search for an existing/missing item in the collection.

Figure 19: Average time for pointwise queries on compressed indexes, built on real-world datasets.



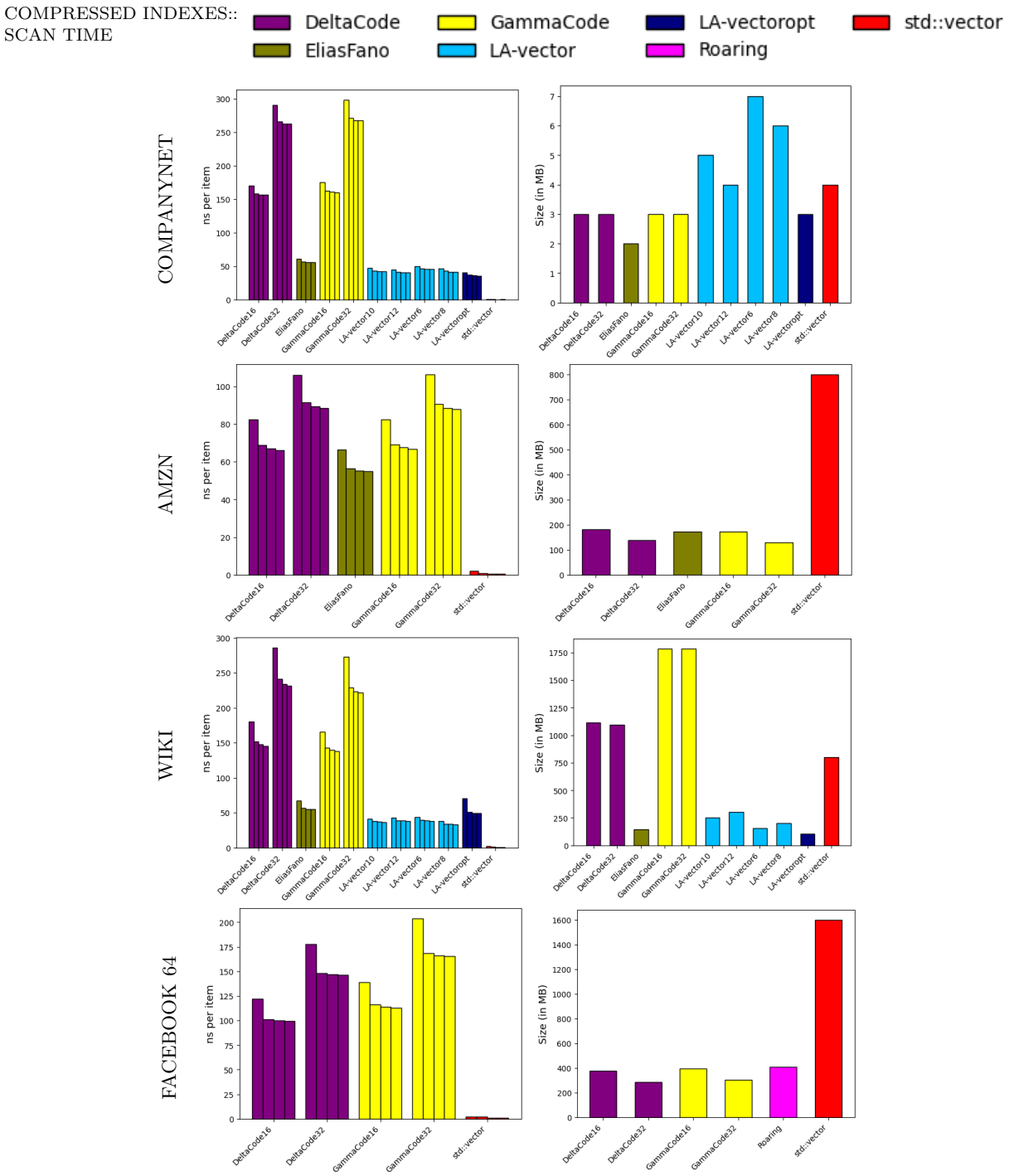
The performances of compressed indexes on synthetic datasets are represented in two plots. The leftmost plot shows the time of search (in ns) on each index; while the right plot shows the space occupied by each index in MBs (without considering the space occupied by the data to be stored). The left plot shows two bars for each data structure. The left/right one shows the average time to search for an existing/missing item in the collection.

Figure 20: Average time for pointwise queries on compressed indexes, built on synthetic datasets.



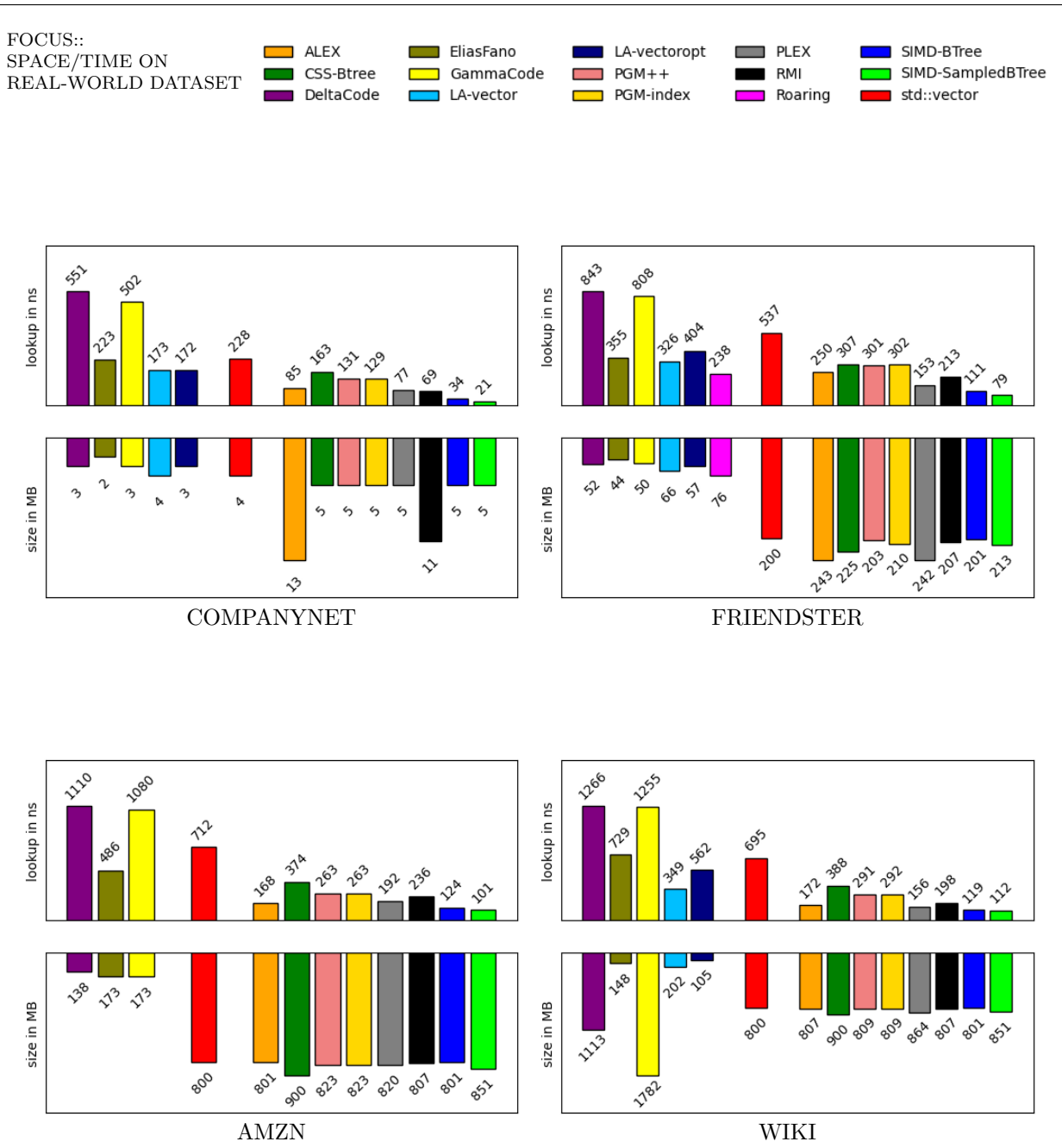
The performances of compressed indexes on 64-bit datasets are represented in two plots. The leftmost plot shows the time of search (in ns) on each index; while the right plot shows the space occupied by each index in MBs (without considering the space occupied by the data to be stored). The left plot shows two bars for each data structure. The left/right one shows the average time to search for an existing/missing item in the collection.

Figure 21: Average time for pointwise queries on compressed indexes, built on 64-bit datasets.



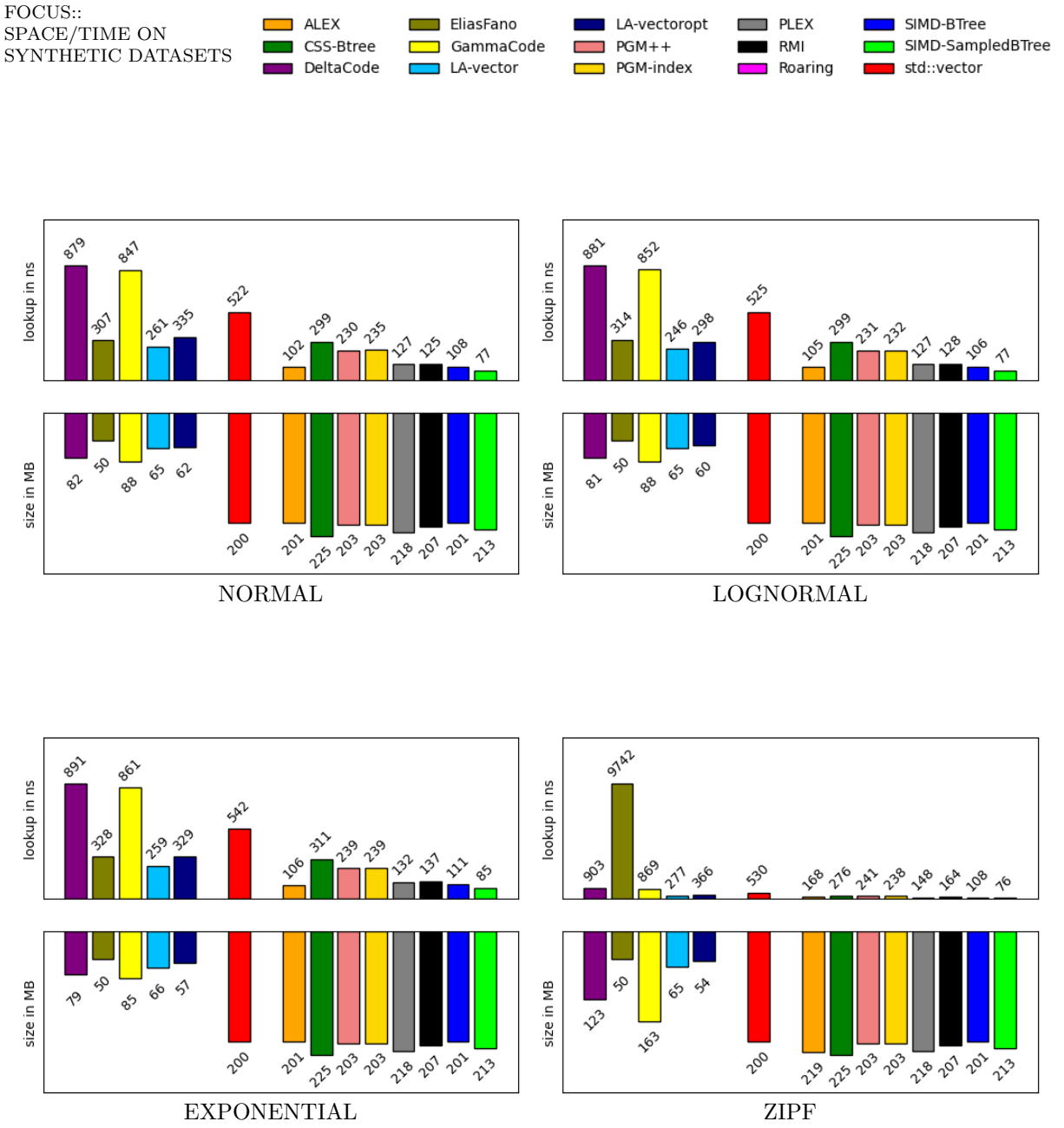
The plots above show the performance of compressed indexes in time and space relative to *range queries*, where starting points are randomly sampled, and the width of the scan is set to 10, 100, 1K, and 10K. The plot on the left shows the time (in ns) required for every interrogation, describing the average time required per access with $x = 10, 100, 1K, 10K$. The plot on the right shows the space occupied by each compressed index (in MB).

Figure 22: Average time (in ns) for range queries on compressed indexes, on 4 real-world datasets.



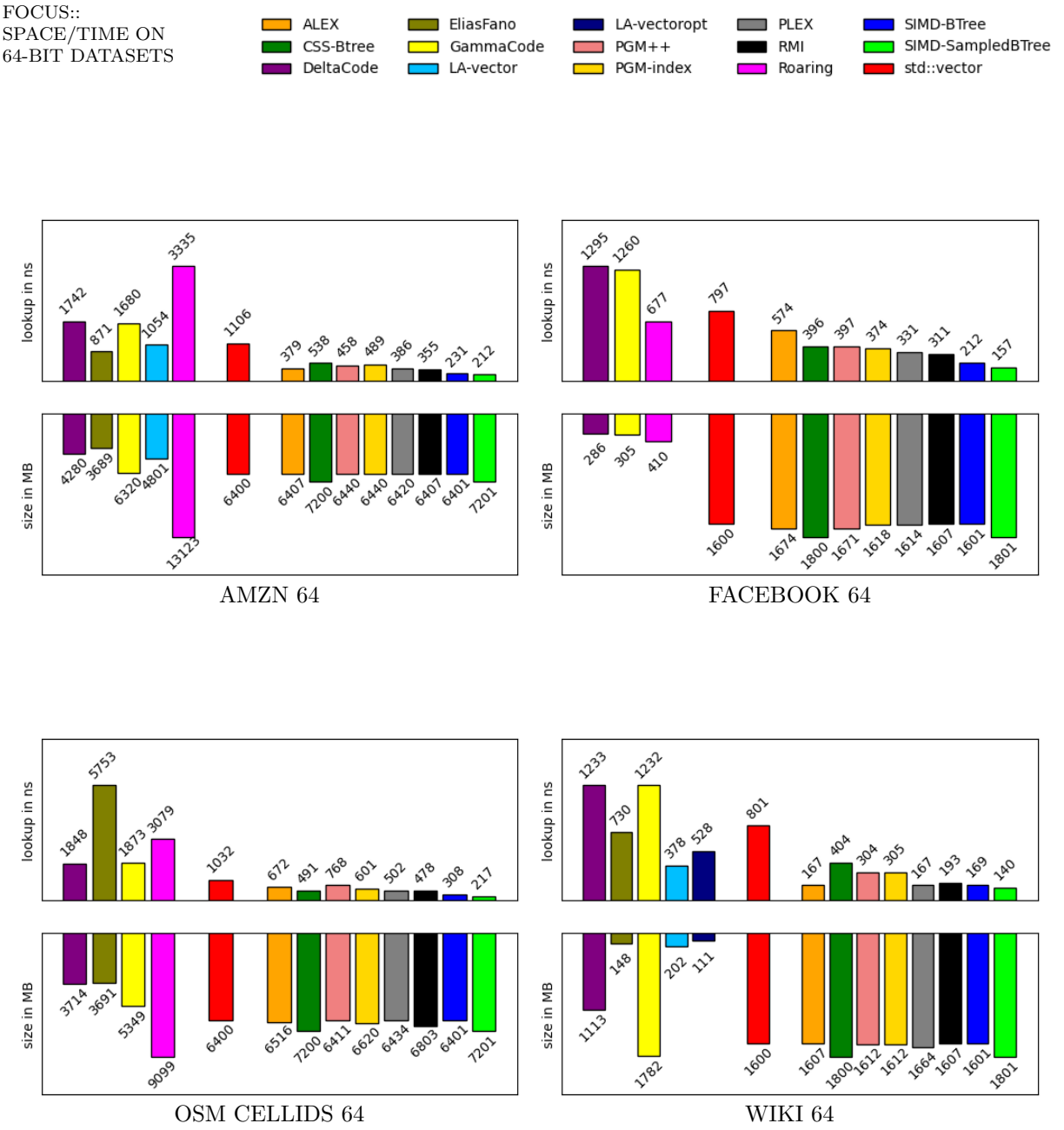
The figures above show the results corresponding to the space occupied and elapsed time for pointwise queries on real-world datasets on all tested indexes (traditional, learned, and compressed). For each dataset and for each index, the top part shows the average time (in ns) needed to make a query on existing items in the dataset, while the bottom part shows the required space in MB (where we added the space of the std::vector for traditional and learned indexes). The results show only the parameter configurations where each index performs best.

Figure 23: Recap space/time plots for pointwise queries on real-world datasets.



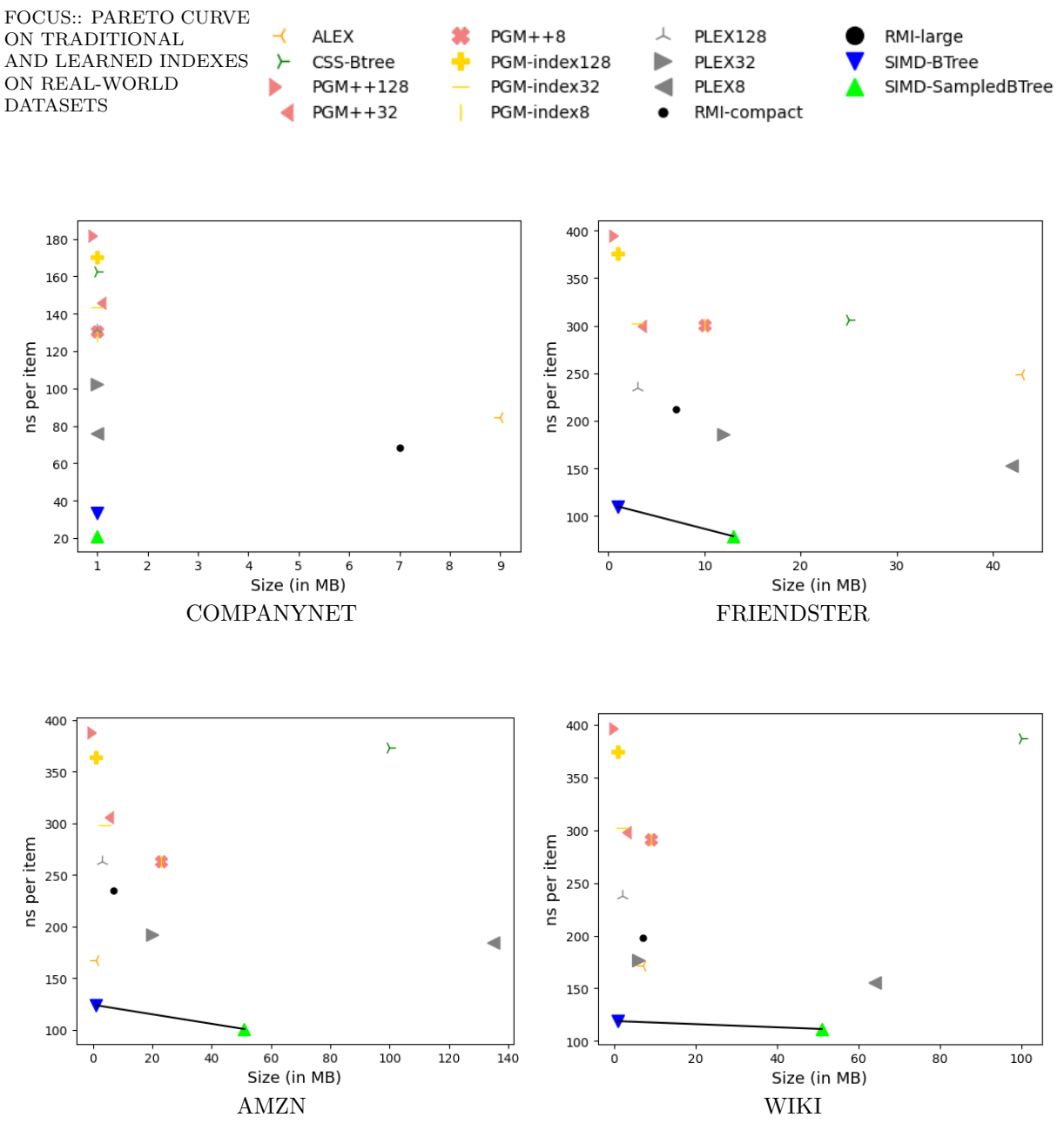
The figures above show the results corresponding to the space occupied and elapsed time for pointwise queries on synthetic datasets on all tested indexes (traditional, learned, and compressed). For each dataset and for each index, the top part shows the average time (in ns) needed to make a query on existing items in the dataset, while the bottom part shows the required space in MB (where we added the space of the std::vector for traditional and learned indexes). The results show only the parameter configurations where each index performs best.

Figure 24: Recap space/time plots for pointwise queries on synthetic datasets.



The figures above show the results corresponding to the space occupied and elapsed time for pointwise queries on 64-bit datasets on all tested indexes (traditional, learned, and compressed). For each dataset and each index, the top part shows the average time (in ns) needed to make a query on existing items in the dataset, while the bottom part shows the required space in MB (where we added the space of the std::vector for traditional and learned indexes). The results show only the parameter configurations where each index performs best.

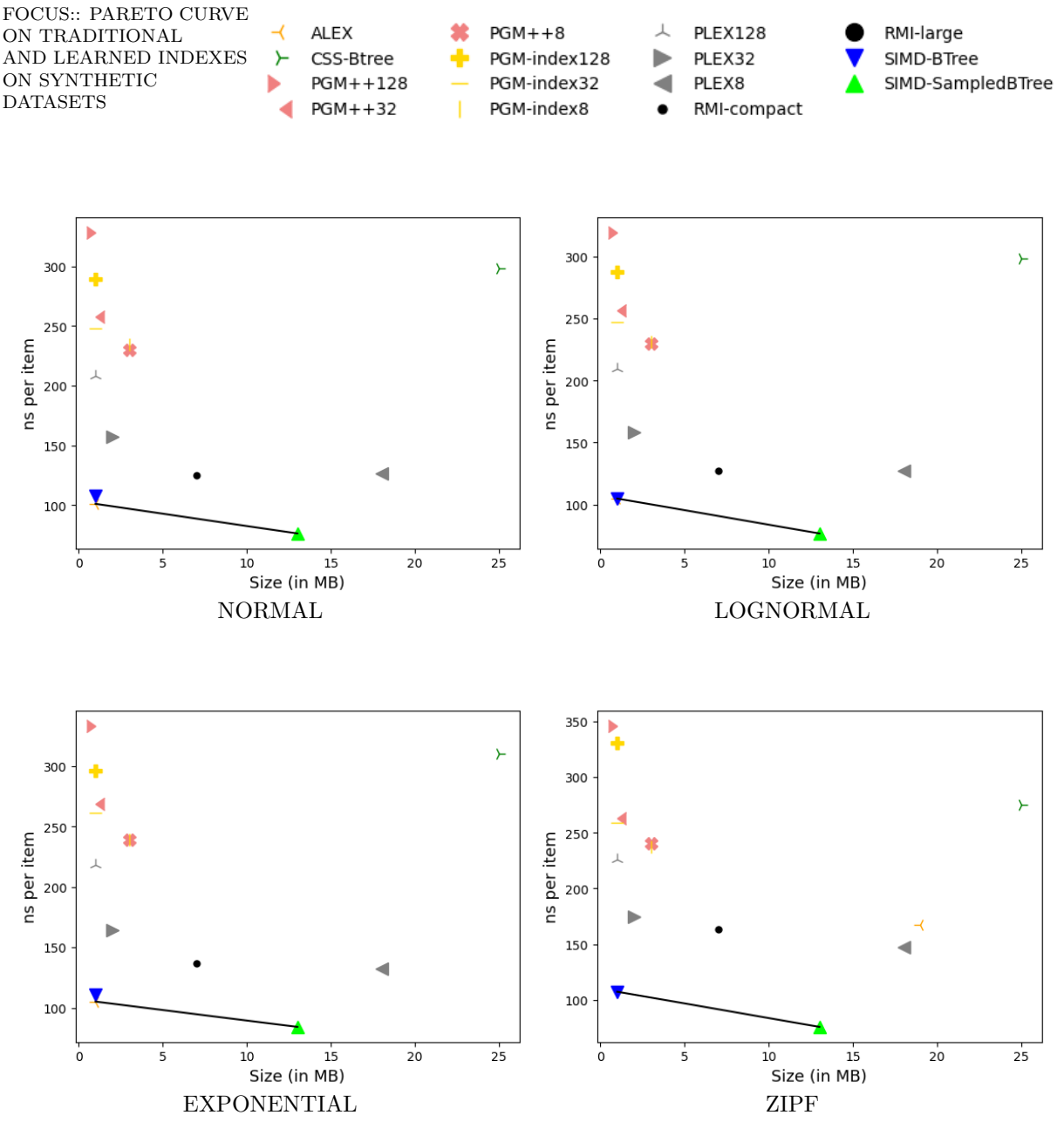
Figure 25: Recap space/time plots for pointwise queries on 64-bit datasets.



The plots recap the experimental results about occupied space/time needed for pointwise queries for traditional and learned indexes on real-world datasets. For these datasets, for each index, and each tested configuration we plot the extra space occupied (in MB) on the x-axis, and the time (in ns) per pointwise query on existing items in the datasets.

Additionally, a black line shows the Pareto frontier for traditional/learned indexes. The indexes that sit on top of the Pareto frontier offer the best space-time trade-off. We avoided plotting "RMI-large" because of its excessive space occupancy (roughly 400 MB on each dataset). "RMI-large" was not one of the Pareto-optimal configurations.

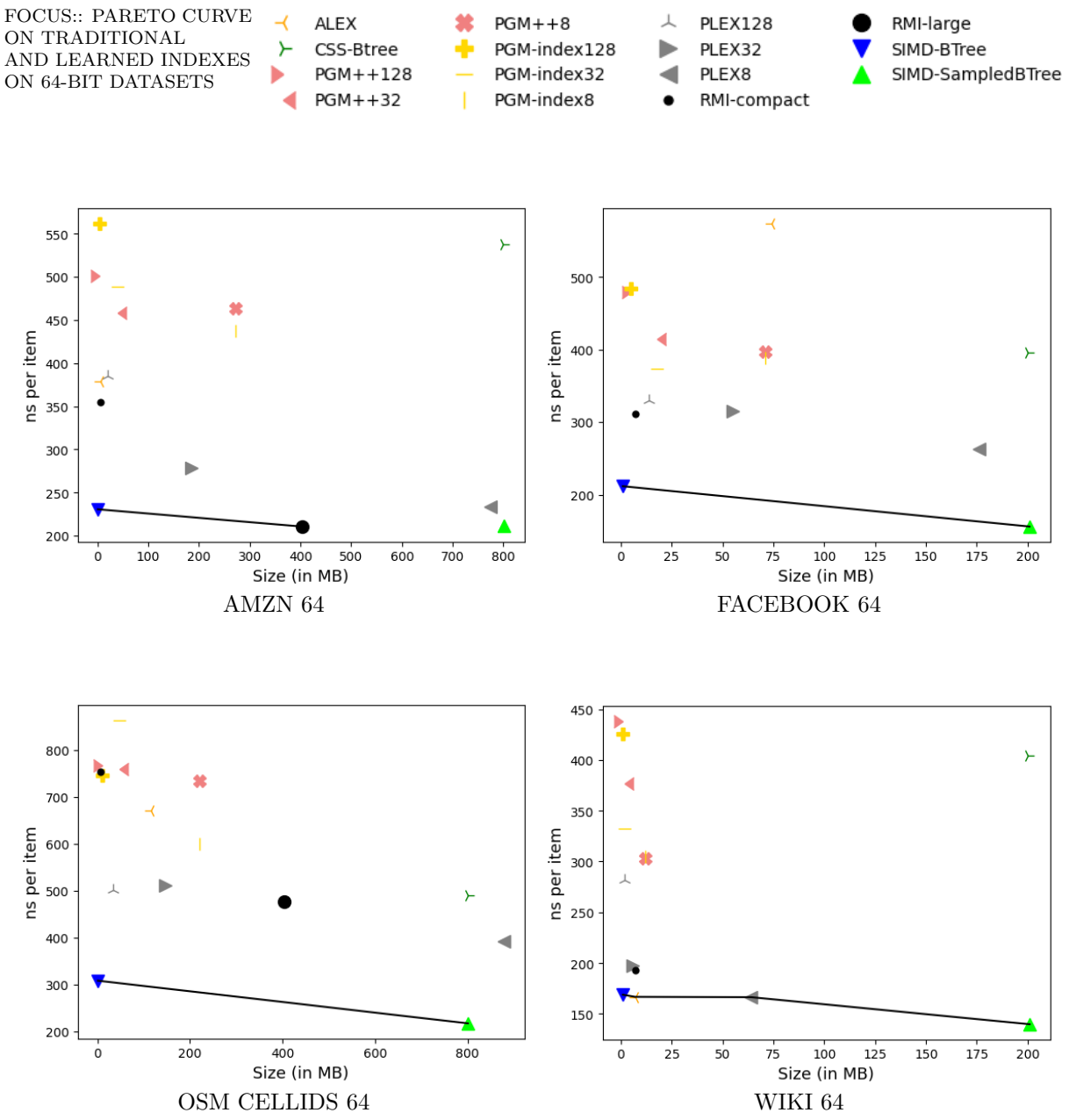
Figure 26: Pareto Frontier: Traditional and Learned Indexes on real-world datasets



The plots recap the experimental results about occupied space/time needed for pointwise queries for traditional and learned indexes on synthetic datasets. For these datasets, for each index, and each tested configuration we plot the extra space occupied (in MB) on the x-axis, and the time (in ns) per pointwise query on existing items in the datasets.

Additionally, a black line shows the Pareto frontier for traditional/learned indexes. The indexes that sit on top of the Pareto frontier offer the best space-time trade-off. We avoided plotting "RMI-large" because of its excessive space occupancy (roughly 400 MB on each dataset). "RMI-large" was not one of the Pareto-optimal configurations.

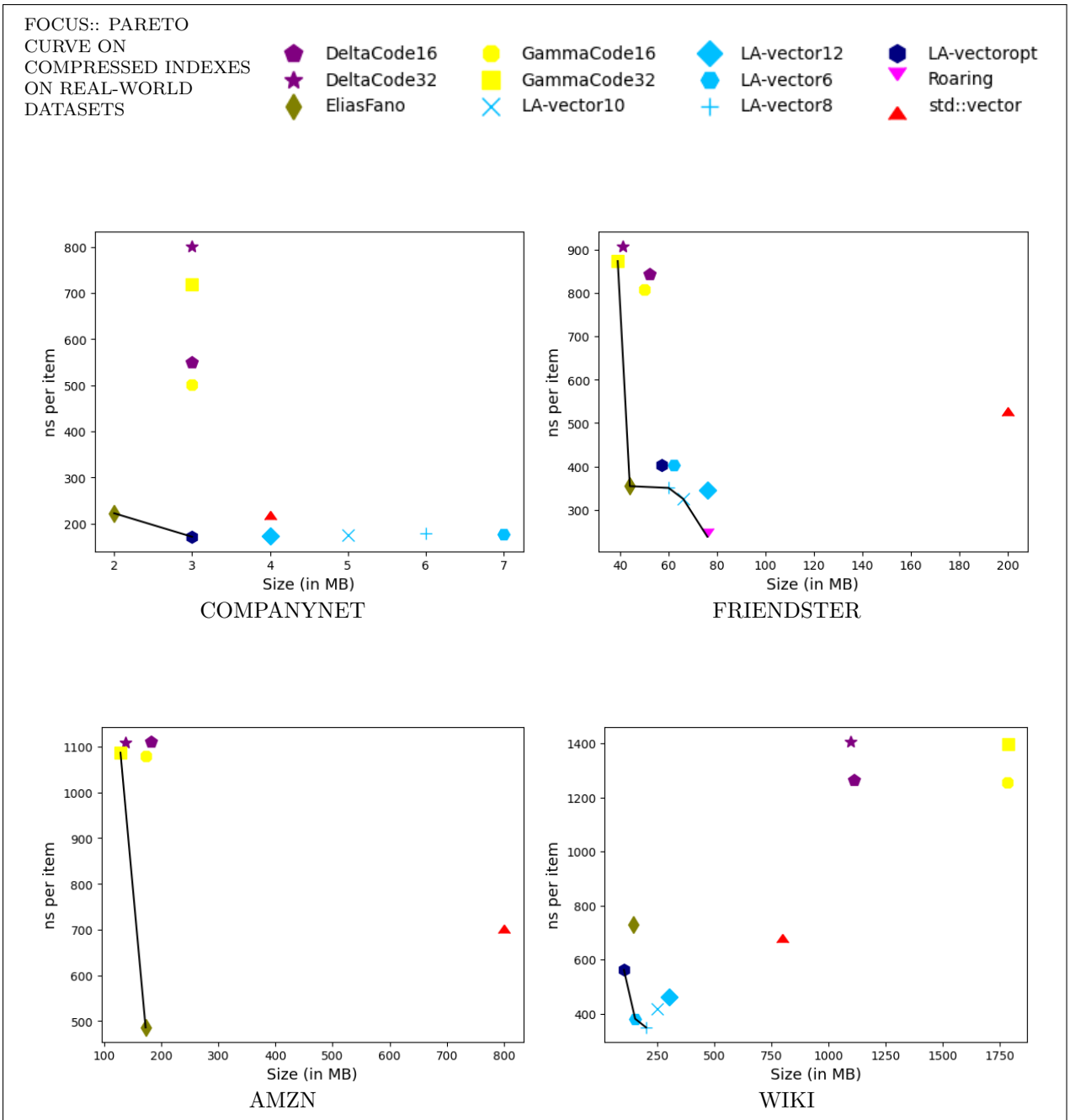
Figure 27: Pareto Frontier: Traditional and Learned Indexes on synthetic datasets



The plots recap the experimental results about occupied space/time needed for pointwise queries for traditional and learned indexes on 64-bit datasets. For these datasets, for each index, and each tested configuration we plot the extra space occupied (in MB) on the x-axis, and the time (in ns) per pointwise query on existing items in the datasets.

Additionally, a black line shows the Pareto frontier for traditional/learned indexes. The indexes that sit on top of the Pareto frontier offer the best space-time trade-off. We avoided plotting "RMI-large" because of its excessive space occupancy (roughly 400 MB on each dataset). "RMI-large" was not one of the Pareto-optimal configurations.

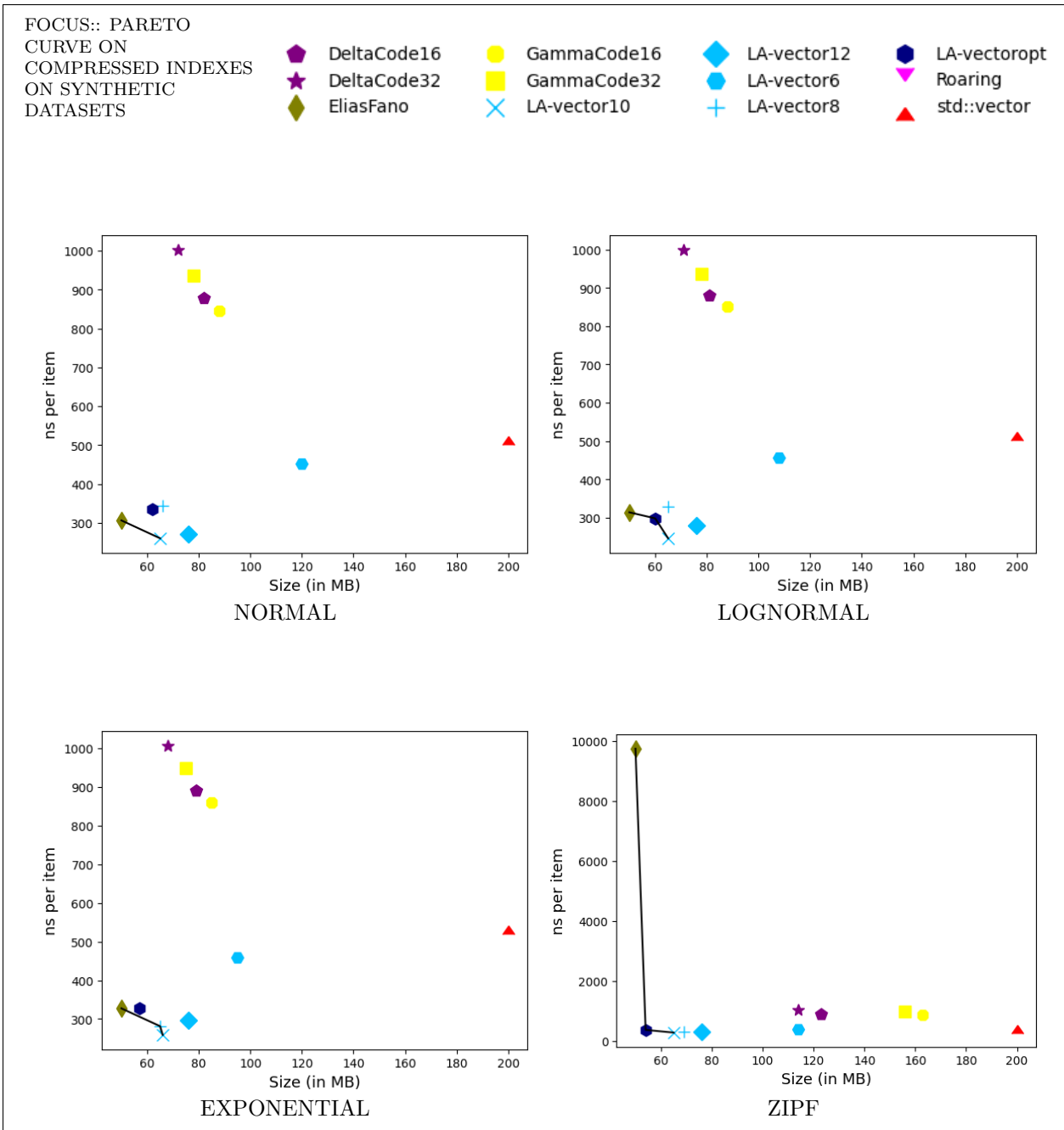
Figure 28: Pareto Frontier: Traditional and Learned Indexes on 64-bit datasets



The plots recap the experimental results about occupied space/time needed for pointwise queries for compressed indexes on real-world datasets. For these datasets, for each compressed index, and each tested configuration we plot the extra space occupied (in MB) on the x-axis, and the time (in ns) per pointwise query on existing items in the datasets.

Additionally, a black line shows the Pareto frontier for traditional/learned indexes. The indexes that sit on top of the Pareto frontier offer the best space-time trade-off.

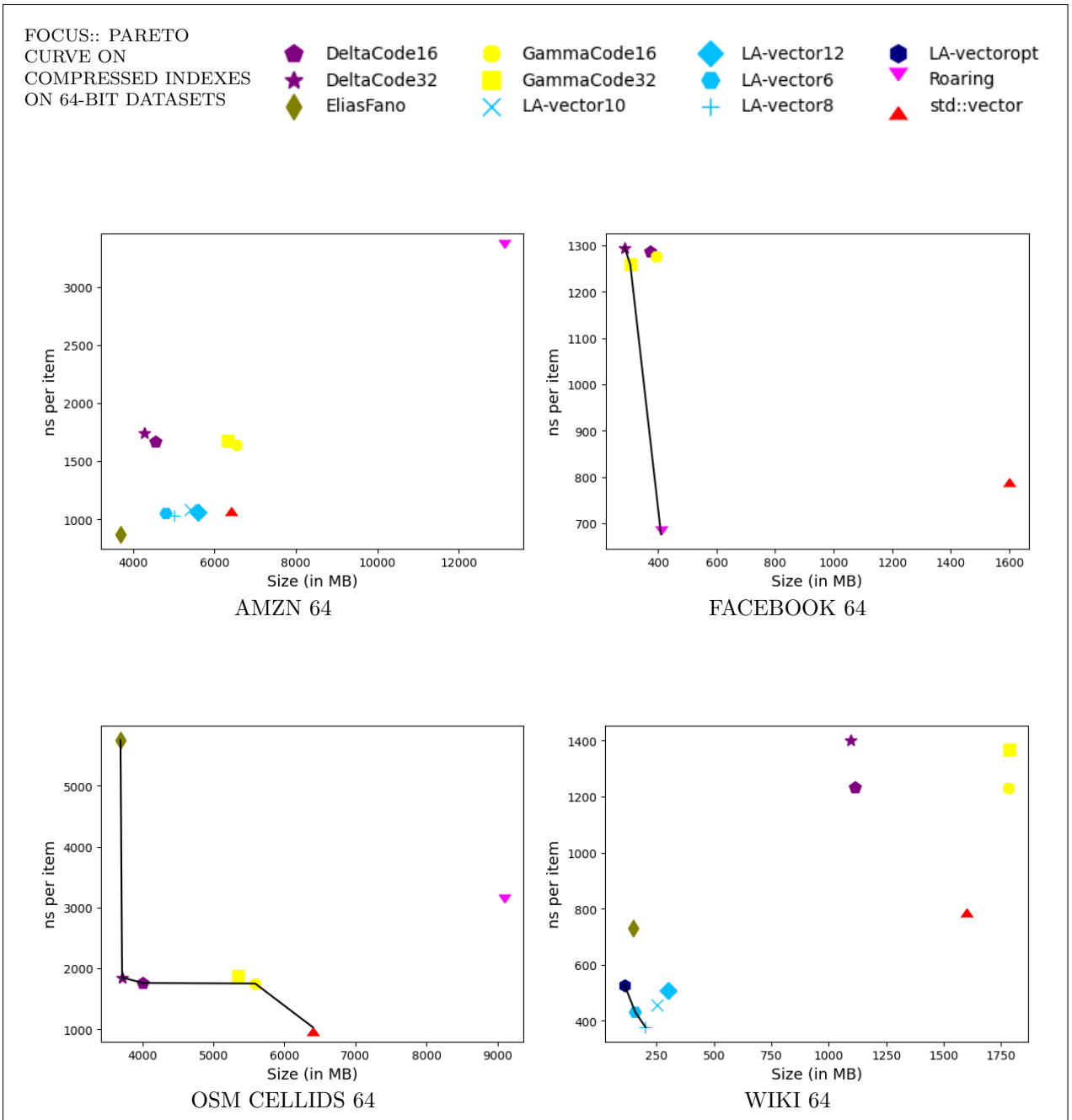
Figure 29: Pareto Frontier: Compressed Indexes on real-world datasets



The plots recap the experimental results about occupied space/time needed for pointwise queries for compressed indexes on synthetic datasets. For these datasets, for each compressed index, and each tested configuration we plot the extra space occupied (in MB) on the x-axis, and the time (in ns) per pointwise query on existing items in the datasets.

Additionally, a black line shows the Pareto frontier for traditional/learned indexes. The indexes that sit on top of the Pareto frontier offer the best space-time trade-off.

Figure 30: Pareto Frontier: Compressed Indexes on synthetic datasets



The plots recap the experimental results about occupied space/time needed for pointwise queries for compressed indexes on 64-bit datasets. For these datasets, for each compressed index, and each tested configuration we plot the extra space occupied (in MB) on the x-axis, and the time (in ns) per pointwise query on existing items in the datasets.

Additionally, a black line shows the Pareto frontier for traditional/learned indexes. The indexes that sit on top of the Pareto frontier offer the best space-time trade-off.

Figure 31: Pareto Frontier: Compressed Indexes on 64-bit datasets

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	84.795	75.2779	9	0.07375703062862157
CSS-Btree	162.639	156.072	1	0.0034730276092886925
DeltaCode16	550.163	550.164	3	0.028494673501700162
DeltaCode32	801.402	801.772	3	0.03199477568268776
EliasFano	222.173	190.123	2	0.034575643856078385
GammaCode16	501.978	497.942	3	0.02185149509459734
GammaCode32	718.739	716.487	3	0.022436260525137187
LA-vector10	174.389	163.978	5	0.04916858002543449
LA-vector12	172.844	158.693	4	0.03182077938690782
LA-vector6	177.356	179.273	7	0.05563622722402215
LA-vector8	177.742	165.176	6	0.05025944449007511
LA-vectoropt	171.206	174.05	3	0.9590857685543597
PGM++128	181.748	182.408	1	0.0023462343961000443
PGM++32	145.778	148.903	1	0.0023386308923363684
PGM++8	130.347	133.17	1	0.003367249108850956
PGM-index128	170.391	176.688	1	0.0023993615992367267
PGM-index32	143.277	144.125	1	0.002358195558190346
PGM-index8	128.272	129.927	1	0.003406153805553913
PLEX128	131.528	141.461	1	0.018869540933519603
PLEX32	102.084	97.9293	1	0.02040441343560815
PLEX8	76.0505	80.3045	1	0.024930240586400032
RMI-compact	68.2063	57.9312	7	0.0031483286991715433
RMI-large	87.104	73.6292	403	0.13805404072627425
SIMD-BTree	33.1707	34.8978	1	0.00420203460380435
SIMD-SampledBTree	20.9854	23.1505	1	0.0028440219350159167
std::vector	227.127	233.441	4	-

Table 1: Tabular data: companynet dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	171.216	130.978	7	12.250281422398983
CSS-Btree	387.456	209.897	100	1.7069255005568267
DeltaCode16	1265.27	766.646	1113	8.585237802937627
DeltaCode32	1406.5	951.687	1095	8.918436351884157
EliasFano	728.747	251.374	148	7.77220363644883
GammaCode16	1254.4	743.391	1782	8.558242926280945
GammaCode32	1395.82	923.777	1785	8.52353121675551
LA-vector10	419.032	237.91	251	1.9248916687443853
LA-vector12	462.707	258.213	301	1.7446172118186951
LA-vector6	380.864	315.315	154	2.069242195412517
LA-vector8	348.619	241.481	202	1.7956738444976508
LA-vectoropt	561.527	285.196	105	197.67387456940486
PGM++128	396.57	256.087	1	0.48669065972790126
PGM++32	298.428	204.902	2	0.48260454786941415
PGM++8	290.848	205.049	9	0.5640171232633293
PGM-index128	374.152	246.121	1	0.4928132425993681
PGM-index32	302.538	201.409	2	0.4936263266019524
PGM-index8	291.528	201.788	9	0.5691779868677258
PLEX128	237.739	174.58	2	3.024151705019176
PLEX32	176.581	121.453	6	3.1280779611319303
PLEX8	155.252	120.38	64	3.7491886020638048
RMI-compact	197.538	106.451	7	0.0028465217910706997
RMI-large	135.004	101.825	403	0.23564787851646543
SIMD-BTree	118.831	71.6056	1	1.5975063968449832
SIMD-SampledBTree	111.367	53.0724	51	1.5690452704206108
std::vector	694.065	360.534	800	-

Table 2: Tabular data: wiki uint32 dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	249.282	228.425	43	3.2466171032749114
CSS-Btree	306.611	287.862	25	0.2028719955123961
DeltaCode16	842.925	838.148	52	1.3043092420324682
DeltaCode32	907.486	935.238	41	1.4376526718959213
EliasFano	354.424	268.447	44	1.888667493313551
GammaCode16	807.403	801.357	50	1.0351957656443118
GammaCode32	873.273	882.188	39	1.0421883559785783
LA-vector10	325.142	276.805	66	0.5263148916885256
LA-vector12	345.036	313.693	76	0.4342422457411885
LA-vector6	402.512	323	62	0.8119859673082829
LA-vector8	350.417	310.847	60	0.5766228409484029
LA-vectoropt	403.264	384.947	57	52.32871983014047
PGM++128	394.567	375.317	1	0.1312911598943174
PGM++32	300.058	288.891	3	0.15784446522593498
PGM++8	300.258	288.922	10	0.20441991137340665
PGM-index128	376.135	365.009	1	0.13260455317795278
PGM-index32	302.368	286.235	3	0.1591664945706725
PGM-index8	301.493	290.429	10	0.20747915282845497
PLEX128	234.662	246.05	3	0.9356957429088653
PLEX32	185.963	165.233	12	1.0998761787079274
PLEX8	152.585	158.389	42	1.4533371008001268
RMI-compact	212.397	177.803	7	0.0017073230817914009
RMI-large	126.462	116.152	403	0.2106458878144622
Roaring	237.821	320.745	76	2.2545253791846336
SIMD-BTree	110.103	104.483	1	0.29207655750215056
SIMD-SampledBTree	78.882	74.924	13	0.16929495614022017
std::vector	536.189	540.293	200	-

Table 3: Tabular data: friendster dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	105.054	85.9853	1	1.7931585018523037
CSS-Btree	310.346	252.282	25	0.20526355598121881
DeltaCode16	890.872	816.098	79	1.375914961565286
DeltaCode32	1006.65	969.474	68	1.5082885845564307
EliasFano	327.316	280.85	50	1.548449839092791
GammaCode16	860.252	798.242	85	1.065692009124905
GammaCode32	949.054	950.769	75	1.0721476834267378
LA-vector10	258.936	234.123	66	0.5412897858768702
LA-vector12	297.423	236.388	76	0.41926944712176917
LA-vector6	459.798	405.767	95	1.1954657504335047
LA-vector8	281.28	303.526	65	0.6488127575255931
LA-vectoropt	328.516	273.581	57	48.586336804553866
PGM++128	333.256	273.656	1	0.08065461004152893
PGM++32	268.757	236.751	1	0.09385824017226696
PGM++8	238.733	211.15	3	0.13035861048847436
PGM-index128	296.382	260.838	1	0.08167381696403027
PGM-index32	261.463	224.831	1	0.09562891321256757
PGM-index8	238.67	207.413	3	0.1327333369292319
PLEX128	218.257	196.585	1	0.8311884633265436
PLEX32	164.447	136.248	2	0.9581081283278763
PLEX8	131.953	121.509	18	1.2444443267770111
RMI-compact	136.908	110.583	7	0.002863059192895889
RMI-large	100.939	101.656	403	0.13079720977693796
SIMD-BTree	110.86	85.3157	1	0.27306970302015543
SIMD-SampledBTree	84.1393	61.7075	13	0.16915717972442507
std::vector	541.996	450.289	200	-

Table 4: Tabular data: exponential dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	104.909	91.3408	1	1.925961231160909
CSS-Btree	298.479	280.769	25	0.21856090221554042
DeltaCode16	880.698	851.485	81	1.4816991727799178
DeltaCode32	999.842	994.071	71	1.6285341411828995
EliasFano	313.658	283.585	50	1.6909992746077478
GammaCode16	851.016	828.575	88	1.1334734314121306
GammaCode32	937.174	953.315	78	1.1604485894553362
LA-vector10	245.619	237.983	65	0.5497058939188719
LA-vector12	279.572	250.127	76	0.43295947359874837
LA-vector6	456.651	425.821	108	1.3747535319067539
LA-vector8	329.223	314.825	65	0.6730958395637572
LA-vectoropt	297.824	307.213	60	51.73580489614979
PGM++128	319.303	272.324	1	0.08486448731273413
PGM++32	256.398	223.029	1	0.10077933352440596
PGM++8	230.012	204.547	3	0.13974548354744912
PGM-index128	287.646	253.085	1	0.08689087992534042
PGM-index32	246.915	219.002	1	0.1028373247012496
PGM-index8	231.244	202.007	3	0.1391121602617204
PLEX128	209.419	209.191	1	0.8810721554793417
PLEX32	158.246	145.083	2	1.0117294962517918
PLEX8	126.922	128.496	18	1.326587322447449
RMI-compact	127.217	118.769	7	0.0032331787049770357
RMI-large	116.657	121.766	403	0.13291828390210866
SIMD-BTree	105.188	104.163	1	0.30701869884505867
SIMD-SampledBTree	76.7902	71.7349	13	0.18232362996786833
std::vector	524.067	488.268	200	-

Table 5: Tabular data: lognormal dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	167.383	175.213	19	3.206527156382799
CSS-Btree	275.09	268.488	25	0.21311991214752196
DeltaCode16	902.359	823.844	123	1.4647299736738204
DeltaCode32	1034.87	966.477	114	1.581426363810897
EliasFano	9741.94	280.261	50	1.5882252238690853
GammaCode16	868.381	802.847	163	1.1896990869194268
GammaCode32	980.971	928.791	156	1.1966758254915475
LA-vector10	276.811	230.765	65	0.5300668471492826
LA-vector12	299.576	233.954	76	0.4063217585906386
LA-vector6	399.53	431.715	114	1.475261174608022
LA-vector8	314.007	332.131	69	0.7096805506385863
LA-vectoropt	365.824	276.092	54	48.09923371775076
PGM++128	345.919	321.539	1	0.08451447850093245
PGM++32	262.665	246.796	1	0.10050335852429271
PGM++8	240.177	224.297	3	0.13639455512166024
PGM-index128	330.631	314.629	1	0.0859683496877551
PGM-index32	258.713	240.903	1	0.10239530261605978
PGM-index8	237.098	221.211	3	0.13921683970838786
PLEX128	226.382	212.227	1	0.795402083452791
PLEX32	174.259	145.882	2	0.9036775778047741
PLEX8	147.392	126.838	18	1.169385984260589
RMI-compact	163.43	110.803	7	0.0019203823059797287
RMI-large	110.573	102.847	403	0.16396219404414297
SIMD-BTree	107.4	88.0679	1	0.2834481427446008
SIMD-SampledBTree	75.8451	64.8493	13	0.18914671139791608
std::vector	529.908	481.796	200	-

Table 6: Tabular data: zipf dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	101.038	89.5688	1	1.5665375479497015
CSS-Btree	298.307	293.062	25	0.2546557548455894
DeltaCode16	878.739	872.093	82	1.3721612793393434
DeltaCode32	1002.53	999.058	72	1.507112545799464
EliasFano	306.291	287.017	50	1.5529780947603284
GammaCode16	846.473	843.143	88	1.0548268908634781
GammaCode32	935.681	944.724	78	1.0782303439453245
LA-vector10	260.203	257.764	65	0.5141205842606724
LA-vector12	271.374	260.617	76	0.39965649181976914
LA-vector6	453.085	443.034	120	1.4047692891210317
LA-vector8	344.367	356.074	66	0.6333029716275632
LA-vectoropt	334.535	324.737	62	48.47124777473509
PGM++128	328.339	323.115	1	0.08510158443823457
PGM++32	258.105	254.79	1	0.10047295140102505
PGM++8	229.672	234.455	3	0.13670747457072138
PGM-index128	289.289	297.185	1	0.08649016143754125
PGM-index32	248.307	244.356	1	0.09907236052677035
PGM-index8	234.847	229.159	3	0.1419549176469445
PLEX128	207.873	215.56	1	0.8297714580781758
PLEX32	157.443	151.75	2	0.9530407358892262
PLEX8	126.653	131.623	18	1.2547215035185217
RMI-compact	124.905	118.324	7	0.0019787837751209735
RMI-large	117.204	120.689	403	0.20049155429005622
SIMD-BTree	107.516	103.903	1	0.273753472790122
SIMD-SampledBTree	76.2583	77.1358	13	0.1795007678680122
std::vector	521.889	521.671	200	-

Table 7: Tabular data: normal dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	167.491	65.8118	1	8.640326638147236
CSS-Btree	373.425	122.897	100	0.8950018980540335
DeltaCode16	1111.35	594.898	182	5.179599394556135
DeltaCode32	1109.42	901.669	138	5.6780403921380636
EliasFano	485.996	263.342	173	8.624733662419022
GammaCode16	1079.56	545.488	173	4.141609657555818
GammaCode32	1086.91	836.371	129	4.184847289696336
PGM++128	387.687	160.025	1	0.4087858649902046
PGM++32	306.026	149.758	4	0.5385268438607455
PGM++8	262.989	136.68	23	0.7104559537954629
PGM-index128	363.67	145.096	1	0.41302481610327957
PGM-index32	297.806	130.935	4	0.5400404484011233
PGM-index8	262.699	124.676	23	0.7184543122537435
PLEX128	263.166	108.248	3	3.356922280602157
PLEX32	191.716	102.956	20	3.736076870933175
PLEX8	184.065	125.158	135	4.980627958942206
RMI-compact	235.018	143.763	7	0.004174442123621702
RMI-large	184.596	134.513	403	0.3163726764731109
SIMD-BTree	123.756	53.5838	1	1.0192036591470242
SIMD-SampledBTree	100.765	41.9515	51	0.6665759796276689
std::vector	711.094	188.601	800	-

Table 8: Tabular data: amzn uint32 dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	378.669	375.659	7	51.12956800432876
CSS-Btree	537.161	473.147	800	13.740855899453162
DeltaCode16	1669.93	1486.91	4554	34.080375880375506
DeltaCode32	1741.05	1641.7	4280	34.1632475846447
EliasFano	870.018	633.128	3689	38.51473539825529
GammaCode16	1641.88	1480.54	6529	29.730070564337073
GammaCode32	1679.7	1529.27	6320	29.658555390592664
LA-vector10	1083.72	930.558	5401	88.64391160393133
LA-vector12	1062.75	852.285	5601	99.87744658198208
LA-vector6	1053.73	879.114	4801	87.54465185552836
LA-vector8	1033.01	825.049	5001	88.54901628801599
PGM++128	501.538	528.537	5	2.04527162918821
PGM++32	457.759	468.919	40	2.536897181160748
PGM++8	463.501	449.182	273	3.7891560167074205
PGM-index128	561.801	490.928	5	2.0934837545268237
PGM-index32	488.42	431.021	40	2.6066456430591645
PGM-index8	437.634	429.747	273	4.060132746119052
PLEX128	385.164	337.635	20	15.909060241281987
PLEX32	278.458	276.138	186	17.986660381499675
PLEX8	233.454	229.739	775	23.171494773123413
RMI-compact	354.264	297.817	7	0.002939328085631132
RMI-large	210.585	193.132	403	0.1930703278630972
Roaring	3334.82	2763.85	13123	240.78884463775904
SIMD-BTree	230.589	213.628	1	13.59278650265187
SIMD-SampledBTree	211.211	198.361	801	13.623581824917347
std::vector	1105.49	1005.29	6400	-

Table 9: Tabular data: amzn uint64 dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	671.084	237.645	116	69.69136697929353
CSS-Btree	490.232	234.792	800	6.9531895454041655
DeltaCode16	1758.77	794.365	4007	29.418319696933033
DeltaCode32	1847.87	1004.56	3714	29.451800709217785
EliasFano	5752.57	1206.97	3691	40.74586051572115
GammaCode16	1749.51	771.32	5589	26.382099382206796
GammaCode32	1872.32	918.599	5349	26.258016065694388
PGM++128	767.681	361.784	11	2.4657752173021437
PGM++32	759.417	310.022	48	2.747436973173171
PGM++8	733.479	302.344	220	3.698034436069429
PGM-index128	745.425	323.391	11	2.4581805650144815
PGM-index32	863.109	338.437	48	2.775276183243841
PGM-index8	600.297	324.233	220	3.746428289171308
PLEX128	501.306	216.359	34	18.645951402187347
PLEX32	510.674	175.989	147	20.487780006974937
PLEX8	390.956	160.757	879	26.167933154292403
RMI-compact	754.173	131.459	7	0.0023391788825392725
RMI-large	477.233	107.309	403	0.1949591820128262
Roaring	3078.13	984.998	9099	187.14552908269687
SIMD-BTree	307.941	148.401	1	8.230164286028593
SIMD-SampledBTree	216.965	109.716	801	6.987790261860937
std::vector	1031.15	497.749	6400	-

Table 10: Tabular data: OSM cellids dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	166.736	146.297	7	13.391475207544865
CSS-Btree	403.955	236.624	200	3.081395957805216
DeltaCode16	1232.16	752.528	1113	9.473718196246773
DeltaCode32	1401.09	941.488	1095	9.415486107673496
EliasFano	729.419	255.153	148	8.688635134417565
GammaCode16	1231.34	752.246	1782	9.50344590730965
GammaCode32	1366.65	921.002	1785	9.39966607633978
LA-vector10	455.316	294.902	251	3.677016740012914
LA-vector12	506.993	296.99	301	3.80449295071885
LA-vector6	431.688	470.686	157	3.6005042964592575
LA-vector8	377.027	237.212	202	3.5147925092838705
LA-vectoropt	527.244	309.344	111	279.10257854238154
PGM++128	438.115	282.648	1	0.5632405459880829
PGM++32	376.922	227.561	2	0.5592115439474583
PGM++8	303.329	232.349	12	0.6455309600569308
PGM-index128	425.689	275.343	1	0.5837437609210611
PGM-index32	332.312	227.73	2	0.575504154805094
PGM-index8	304.436	218.545	12	0.6743493216112256
PLEX128	281.411	195.393	2	3.419378320034593
PLEX32	197.446	143.372	6	3.5555342763662336
PLEX8	166.412	123.024	64	4.248385151289403
RMI-compact	192.971	111.758	7	0.003774173930287361
RMI-large	140.415	106.22	403	0.2906618959270418
SIMD-BTree	168.845	96.0625	1	2.9834817758761343
SIMD-SampledBTree	139.777	83.8916	201	3.248167170304805
std::vector	800.183	396.749	1600	-

Table 11: Tabular data: wiki uint64 dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	573.6	14.5306	74	18.045232778880745
CSS-Btree	395.47	70.4056	200	1.3108691022731365
DeltaCode16	1287.61	414.694	374	6.417910574562848
DeltaCode32	1294.78	527.001	286	6.509404008742422
GammaCode16	1275.4	377.475	393	5.320555119868368
GammaCode32	1259.13	461.428	305	5.343314459919929
PGM++128	479.12	95.6022	5	0.6342772550880909
PGM++32	414.727	89.9523	18	0.7303224395960569
PGM++8	396.81	113.925	71	1.00354130724445
PGM-index128	484.534	96.599	5	0.6531588993966579
PGM-index32	373.327	94.9386	18	0.7532375046052039
PGM-index8	388.017	101.629	71	1.0239377717487514
PLEX128	330.364	61.1977	14	4.366793395299465
PLEX32	315.18	49.0181	55	4.848923907708376
PLEX8	262.922	52.3193	176	6.190323376934976
RMI-compact	310.753	37.2934	7	0.0023477911949157716
RMI-large	213.143	37.4332	403	0.20778175862506032
Roaring	676.09	44.9812	410	22.295894460659472
SIMD-BTree	211.628	50.3897	1	1.318875862658024
SIMD-SampledBTree	156.053	33.8489	201	1.299439326580614
std::vector	796.749	72.0668	1600	-

Table 12: Tabular data: FB uint64 dataset

Compressed Index	Scan 10	Scan 100	Scan 1K	Scan 10K
DeltaCode16	169.902	158.382	156.902	156.262
DeltaCode32	290.359	266.085	262.788	262.641
EliasFano	60.6795	56.5687	56.0174	55.8601
GammaCode16	174.914	162.492	160.746	160.167
GammaCode32	298.385	271.448	268.043	267.507
LA-vector10	47.289	43.3315	42.4952	42.3297
LA-vector12	45.0734	41.4511	40.76	40.6897
LA-vector6	49.6115	46.3757	45.3586	45.2125
LA-vector8	46.0287	42.6529	41.6661	41.5467
LA-vectoropt	40.5808	36.8136	35.9605	35.7708
std::vector	0.562015	0.220738	0.175476	0.251958

Table 13: Tabular data: SCAN experiments on the companynet dataset (times are expressed in ns)

Compressed Index	Scan 10	Scan 100	Scan 1K	Scan 10K
DeltaCode16	180.626	152.231	147.45	145.791
DeltaCode32	286.332	241.294	233.723	231.971
EliasFano	67.0721	56.6032	55.4092	55.0058
GammaCode16	165.895	143.288	139.686	137.795
GammaCode32	272.873	229.1	223.36	221.512
LA-vector10	40.9013	38.0294	37.3074	36.7726
LA-vector12	42.8927	39.0475	39.1278	38.3544
LA-vector6	43.78	39.3341	38.73	38.2743
LA-vector8	38.0936	34.117	34.0408	33.5231
LA-vectoropt	70.3335	51.1309	49.5432	49.0797
std::vector	2.12109	1.38943	0.51784	0.472924

Table 14: Tabular data: SCAN experiments on the wiki uint32 dataset (times are expressed in ns)

Compressed Index	Scan 10	Scan 100	Scan 1K	Scan 10K
DeltaCode16	82.4027	68.7572	67.0358	66.135
DeltaCode32	105.84	91.3301	89.1847	88.4595
EliasFano	66.3994	56.3723	55.3097	54.8652
GammaCode16	82.4171	68.9434	67.5749	66.7945
GammaCode32	106.273	90.424	88.482	87.7333
std::vector	2.05005	0.72798	0.505714	0.435934

Table 15: Tabular data: SCAN experiments on the amzn uint32 dataset (times are expressed in ns)

Compressed Index	Scan 10	Scan 100	Scan 1K	Scan 10K
DeltaCode16	122.292	100.978	100.201	99.5831
DeltaCode32	177.565	147.963	147.04	146.48
GammaCode16	138.96	116.163	113.737	112.854
GammaCode32	203.884	168.636	165.93	165.224
std::vector	2.35548	2.16368	0.963471	0.925722

Table 16: Tabular data: SCAN experiments on the FB uint64 dataset (times are expressed in ns)

Memory Footprint

ALEX: MAX: 10.71x (companynet_uint32) - MIN: 5.63x (wiki_ts_200M_uint64)
Roaring: MAX: 19.37x (books_800M_uint64) - MIN: 2.01x (wiki_ts_200M_uint64)
EliasFano: MAX: 2.61x (fb_200M_uint64) - MIN: 1.86x (companynet_uint32)
GammaCode16: MAX: 4.21x (wiki_ts_200M_uint32) - MIN: 2.00x (companynet_uint32)
GammaCode32: MAX: 4.22x (wiki_ts_200M_uint32) - MIN: 2.00x (companynet_uint32)
DeltaCode16: MAX: 3.38x (wiki_ts_200M_uint32) - MIN: 1.86x (companynet_uint32)
DeltaCode32: MAX: 4.22x (wiki_ts_200M_uint32) - MIN: 2.00x (companynet_uint32)
LA-vector6: MAX: 11.38x (osm_cellids_800M_uint64) - MIN: 2.11x (wiki_ts_200M_uint64)
LA-vector8: MAX: 11.11x (osm_cellids_800M_uint64) - MIN: 2.13x (wiki_ts_200M_uint64)
LA-vector10: MAX: 11.06x (osm_cellids_800M_uint64) - MIN: 2.16x (wiki_ts_200M_uint64)
LA-vector12: MAX: 15.51x (osm_cellids_800M_uint64) - MIN: 2.19x (wiki_ts_200M_uint64)
CSS-BTree: MAX: 3.25x (books_800M_uint64) - MIN: 2.29x (companynet_uint32)
PLEX8: MAX: 1.64x (friendster_50M_uint32) - MIN: 1.10x (wiki_ts_200M_uint64)
PLEX32: MAX: 1.29x (companynet_uint32) - MIN: 1.01x (wiki_ts_200M_uint64)
PLEX128: MAX: 1.29x (companynet_uint32) - MIN: 1.00x (wiki_ts_200M_uint64)
PGM-index8: MAX: 2.30x (friendster_50M_uint32) - MIN: 1.71x (companynet_uint32)
PGM-index32: MAX: 2.07x (friendster_50M_uint32) - MIN: 1.71x (companynet_uint32)
PGM-index128: MAX: 2.02x (fb_200M_uint64) - MIN: 1.71x (companynet_uint32)
PGM++8: MAX: 2.30x (friendster_50M_uint32) - MIN: 1.71x (companynet_uint32)
PGM++32: MAX: 2.07x (friendster_50M_uint32) - MIN: 1.71x (companynet_uint32)
PGM++128: MAX: 2.02x (fb_200M_uint64) - MIN: 1.71x (companynet_uint32)
SIMD-BTree: MAX: 3.48x (books_800M_uint64) - MIN: 2.14x (companynet_uint32)
SIMD-SampledBTree: MAX: 3.30x (books_800M_uint64) - MIN: 2.29x (companynet_uint32)
LA-vectoropt: MAX: 101.87x (fb_200M_uint64) - MIN: 10.57x (companynet_uint32)

Error Report

LA-vectoropt - fb_200M_uint64 - existing: First correction too large
LA-vector6 - fb_200M_uint64 - existing: Bit fields' sizes are not large enough
LA-vector8 - fb_200M_uint64 - existing: Bit fields' sizes are not large enough
LA-vector10 - fb_200M_uint64 - existing: Bit fields' sizes are not large enough
LA-vector12 - fb_200M_uint64 - existing: Bit fields' sizes are not large enough
LA-vectoropt - books_800M_uint64 - existing: First correction too large
LA-vectoropt - osm_cellids_800M_uint64 - existing: First correction too large
LA-vector6 - osm_cellids_800M_uint64 - existing: Segment correction too large
LA-vector8 - osm_cellids_800M_uint64 - existing: Segment correction too large
LA-vector10 - osm_cellids_800M_uint64 - existing: Segment correction too large
LA-vector12 - osm_cellids_800M_uint64 - existing: Segment correction too large
LA-vectoropt - books_200M_uint32 - existing: Bit fields' sizes are not large enough
LA-vector6 - books_200M_uint32 - existing: Bit fields' sizes are not large enough
LA-vector8 - books_200M_uint32 - existing: Bit fields' sizes are not large enough
LA-vector10 - books_200M_uint32 - existing: Bit fields' sizes are not large enough
LA-vector12 - books_200M_uint32 - existing: Bit fields' sizes are not large enough
LA-vectoropt - fb_200M_uint64 - missing: First correction too large
LA-vector6 - fb_200M_uint64 - missing: Bit fields' sizes are not large enough
LA-vector8 - fb_200M_uint64 - missing: Bit fields' sizes are not large enough
LA-vector10 - fb_200M_uint64 - missing: Bit fields' sizes are not large enough
LA-vector12 - fb_200M_uint64 - missing: Bit fields' sizes are not large enough
LA-vectoropt - books_800M_uint64 - missing: First correction too large
LA-vectoropt - osm_cellids_800M_uint64 - missing: First correction too large

```
LA-vector6 - osm_cellids_800M_uint64 - missing: Segment correction too large
LA-vector8 - osm_cellids_800M_uint64 - missing: Segment correction too large
LA-vector10 - osm_cellids_800M_uint64 - missing: Segment correction too large
LA-vector12 - osm_cellids_800M_uint64 - missing: Segment correction too large
LA-vectoropt - books_200M_uint32 - missing: Bit fields' sizes are not large enough
LA-vector6 - books_200M_uint32 - missing: Bit fields' sizes are not large enough
LA-vector8 - books_200M_uint32 - missing: Bit fields' sizes are not large enough
LA-vector10 - books_200M_uint32 - missing: Bit fields' sizes are not large enough
LA-vector12 - books_200M_uint32 - missing: Bit fields' sizes are not large enough
LA-vectoropt - fb_200M_uint64 - buildtime: First correction too large
LA-vector6 - fb_200M_uint64 - buildtime: Bit fields' sizes are not large enough
LA-vector8 - fb_200M_uint64 - buildtime: Bit fields' sizes are not large enough
LA-vector10 - fb_200M_uint64 - buildtime: Bit fields' sizes are not large enough
LA-vector12 - fb_200M_uint64 - buildtime: Bit fields' sizes are not large enough
LA-vectoropt - books_800M_uint64 - buildtime: First correction too large
LA-vectoropt - osm_cellids_800M_uint64 - buildtime: First correction too large
LA-vector6 - osm_cellids_800M_uint64 - buildtime: Segment correction too large
LA-vector8 - osm_cellids_800M_uint64 - buildtime: Segment correction too large
LA-vector10 - osm_cellids_800M_uint64 - buildtime: Segment correction too large
LA-vector12 - osm_cellids_800M_uint64 - buildtime: Segment correction too large
LA-vectoropt - books_200M_uint32 - buildtime: Bit fields' sizes are not large enough
LA-vector6 - books_200M_uint32 - buildtime: Bit fields' sizes are not large enough
LA-vector8 - books_200M_uint32 - buildtime: Bit fields' sizes are not large enough
LA-vector10 - books_200M_uint32 - buildtime: Bit fields' sizes are not large enough
LA-vector12 - books_200M_uint32 - buildtime: Bit fields' sizes are not large enough
LA-vectoropt - fb_200M_uint64 - scan: First correction too large
LA-vector6 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vector8 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vector10 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vector12 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vectoropt - fb_200M_uint64 - scan: First correction too large
LA-vector6 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vector8 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vector10 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vector12 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vectoropt - fb_200M_uint64 - scan: First correction too large
LA-vector6 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vector8 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vector10 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vector12 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vectoropt - fb_200M_uint64 - scan: First correction too large
LA-vector6 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vector8 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vector10 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vector12 - fb_200M_uint64 - scan: Bit fields' sizes are not large enough
LA-vectoropt - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector6 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector8 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector10 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector12 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vectoropt - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector6 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
```

```
LA-vector8 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector10 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector12 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vectoropt - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector6 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector8 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector10 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector12 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vectoropt - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector6 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector8 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector10 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
LA-vector12 - books_200M_uint32 - scan: Bit fields' sizes are not large enough
```