REAL-WORLD
DATASETS DISTRIBUTION

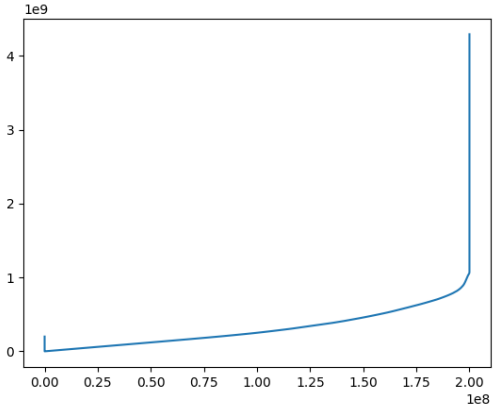


Figure 1: Amzn uint32

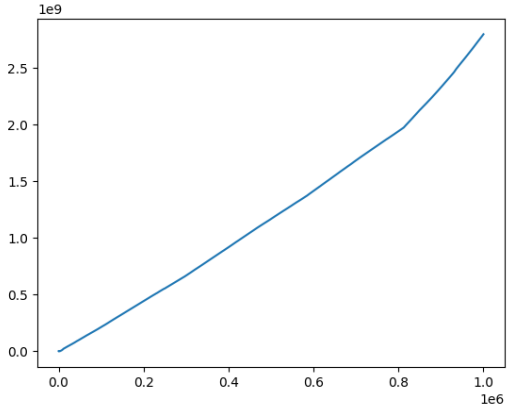


Figure 2: CompanyNet

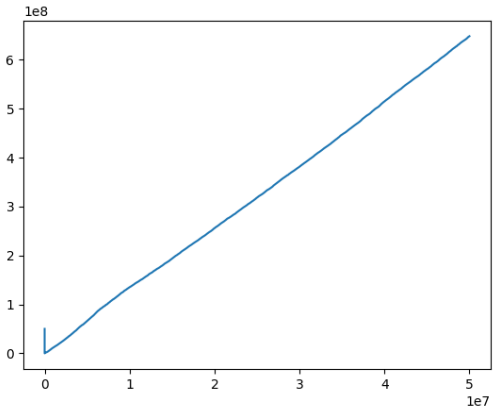


Figure 3: Friendster

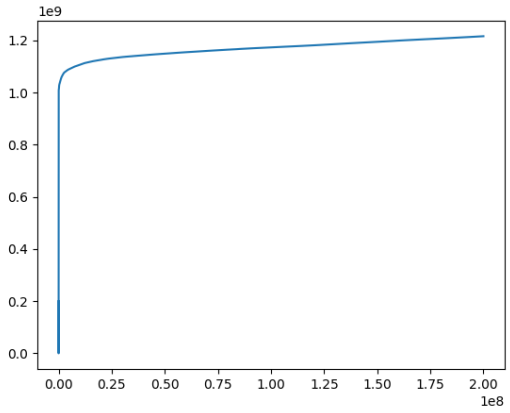


Figure 4: Wiki uint32

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

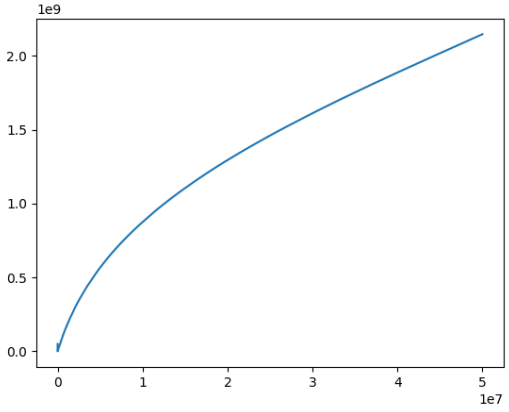


Figure 5: Normal

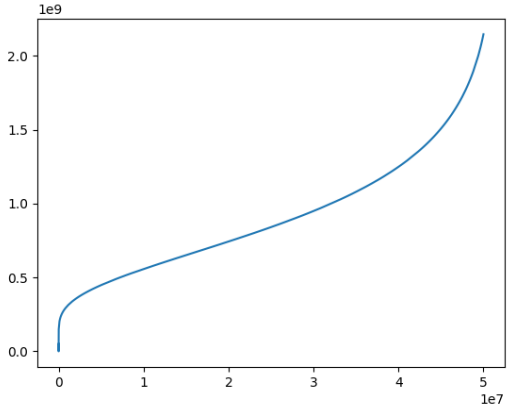


Figure 6: Lognormal

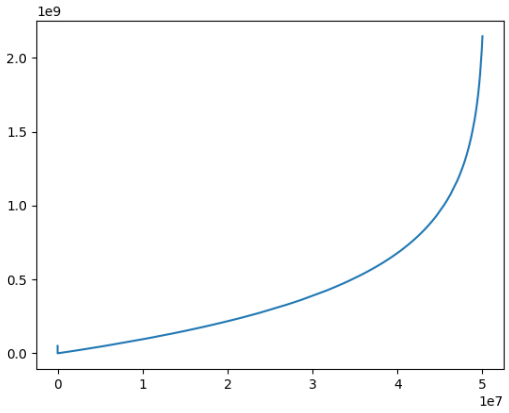


Figure 7: Exponential

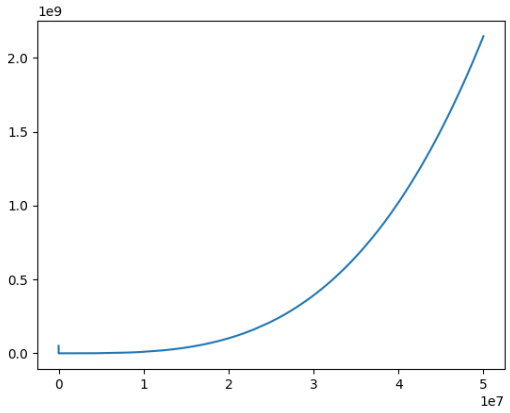


Figure 8: Zipf

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

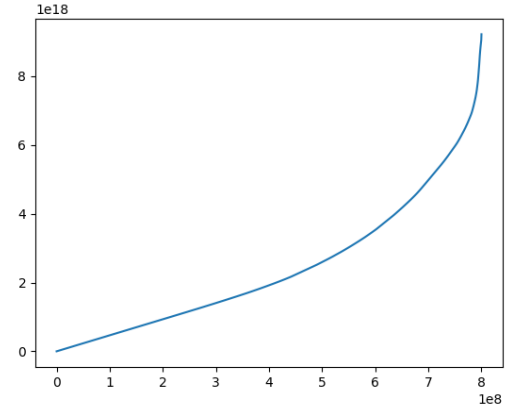


Figure 9: Amzn uint64

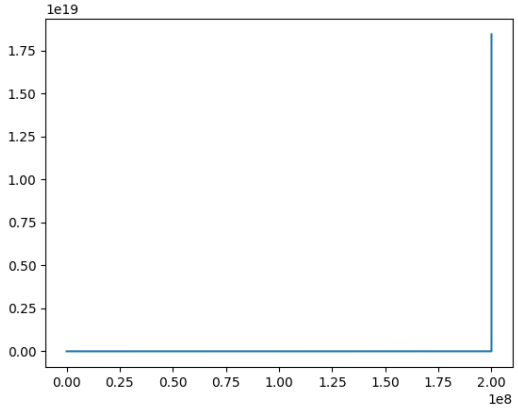


Figure 10: Facebook uint64

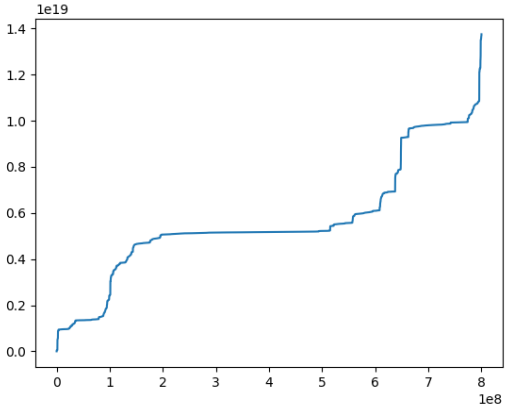


Figure 11: OSM Cellids uint64

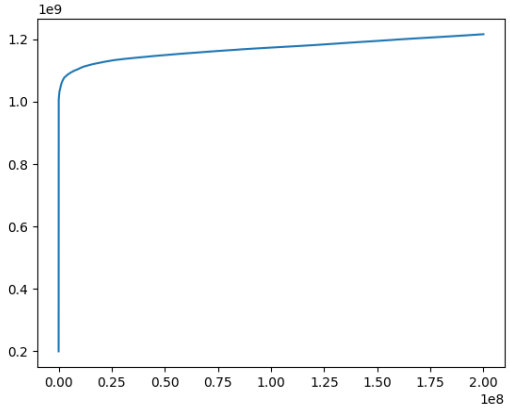
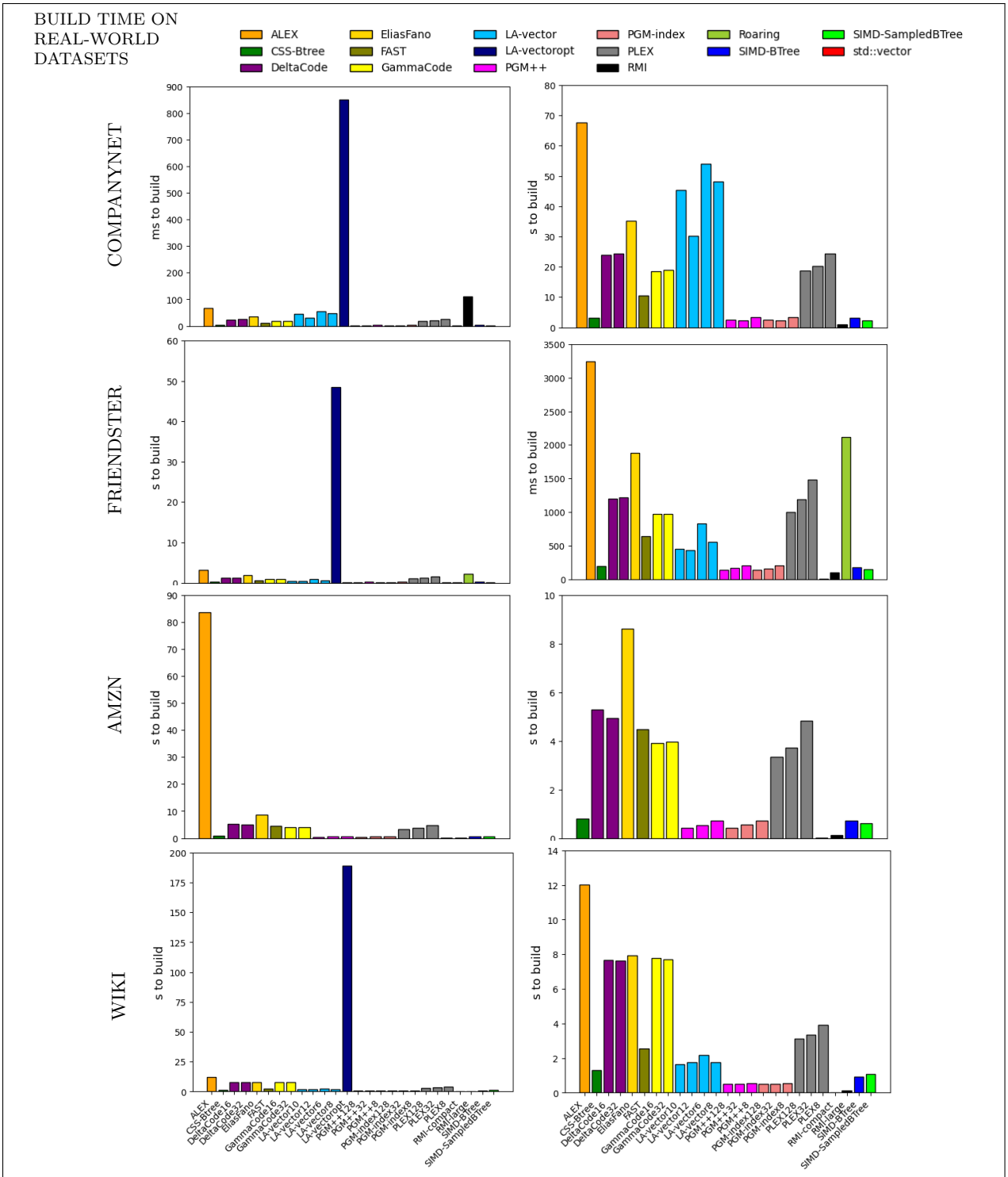


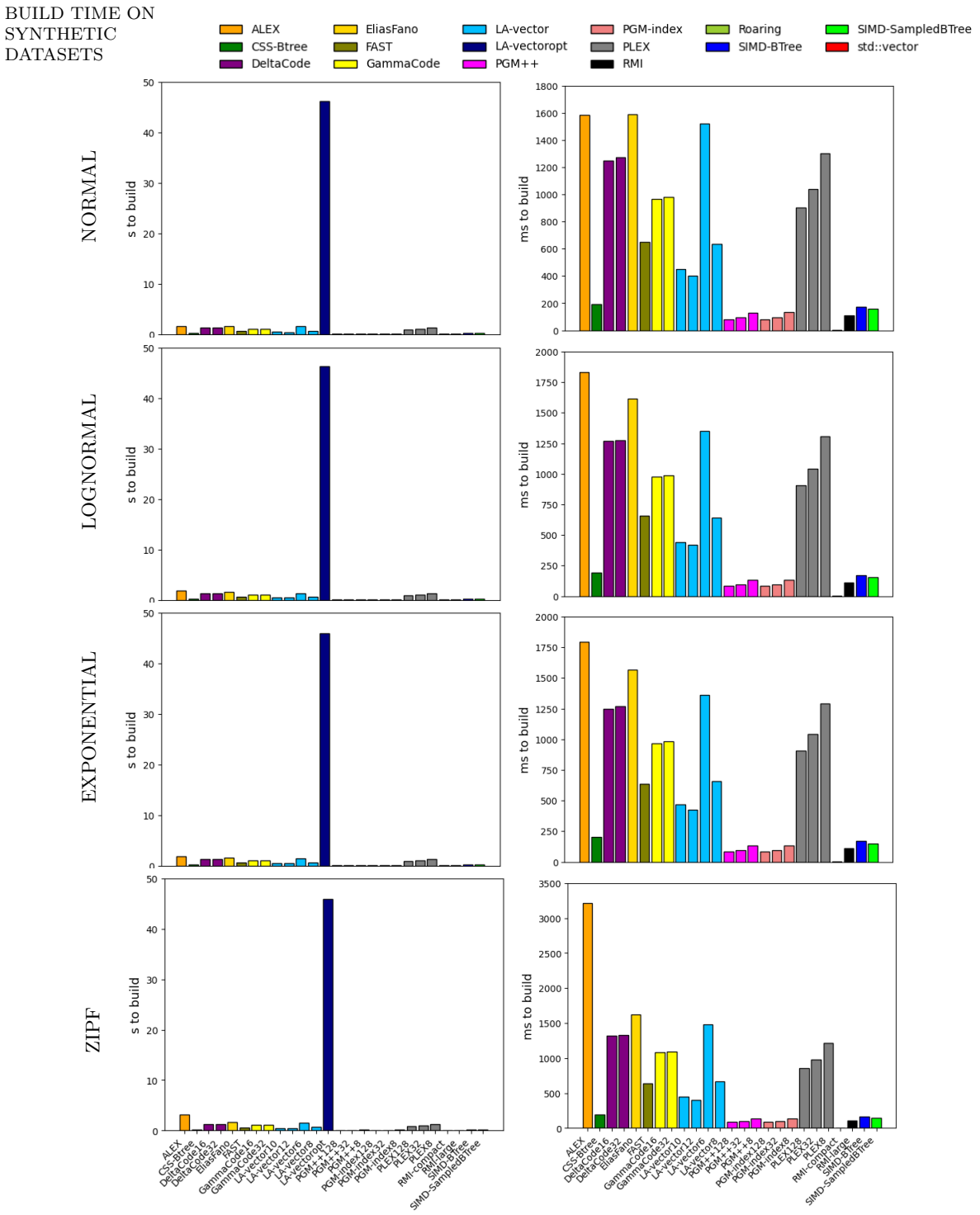
Figure 12: Wiki uint64

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.



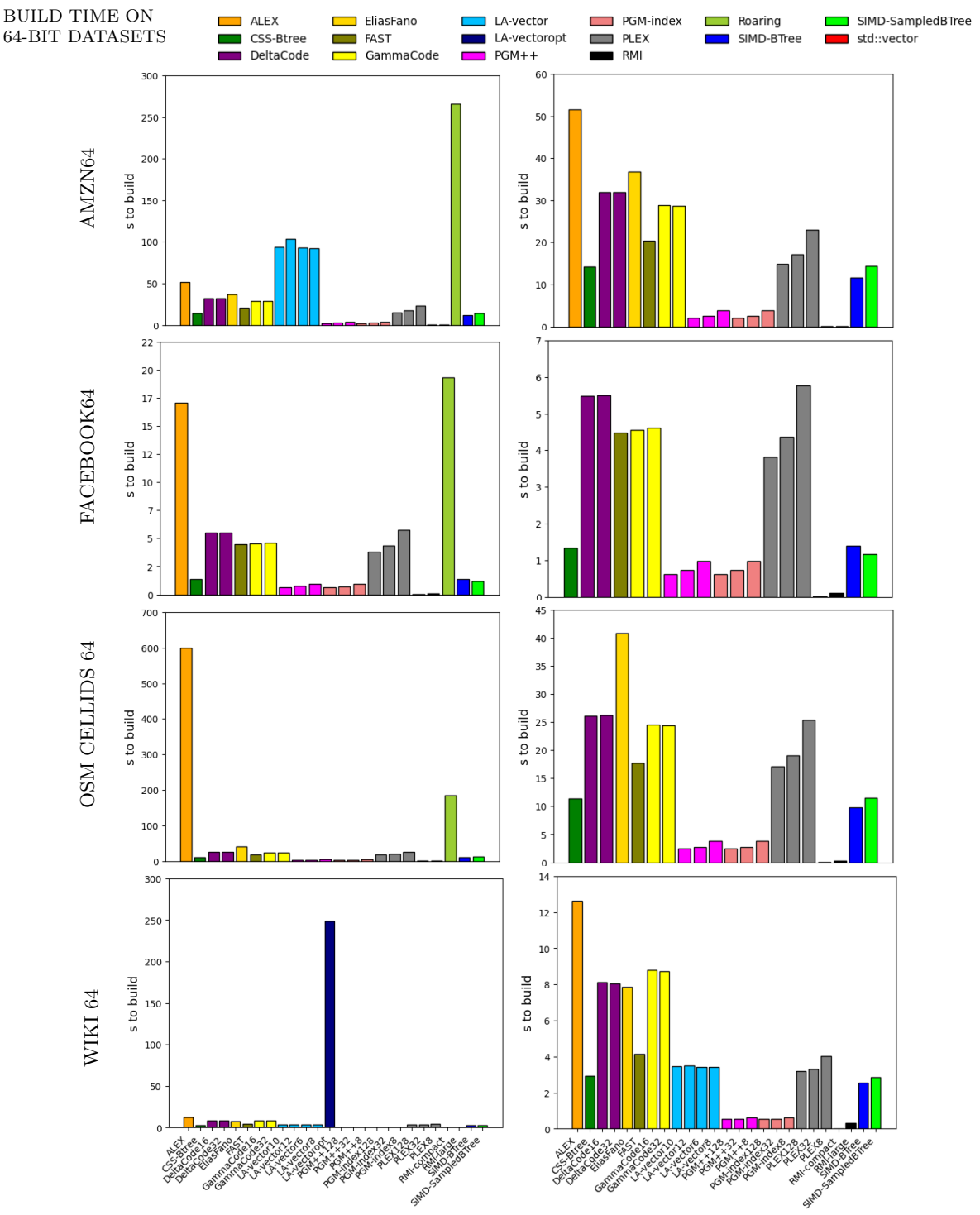
The plots above report the time required to build the data structures (indexes) on the real-world datasets. On the left, the dataset's name is shown; in the middle, the complete plot with all the times expressed in milliseconds; and on the right, the same plot but excluding the ones with excessive build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

Figure 13: Build time on real-world datasets



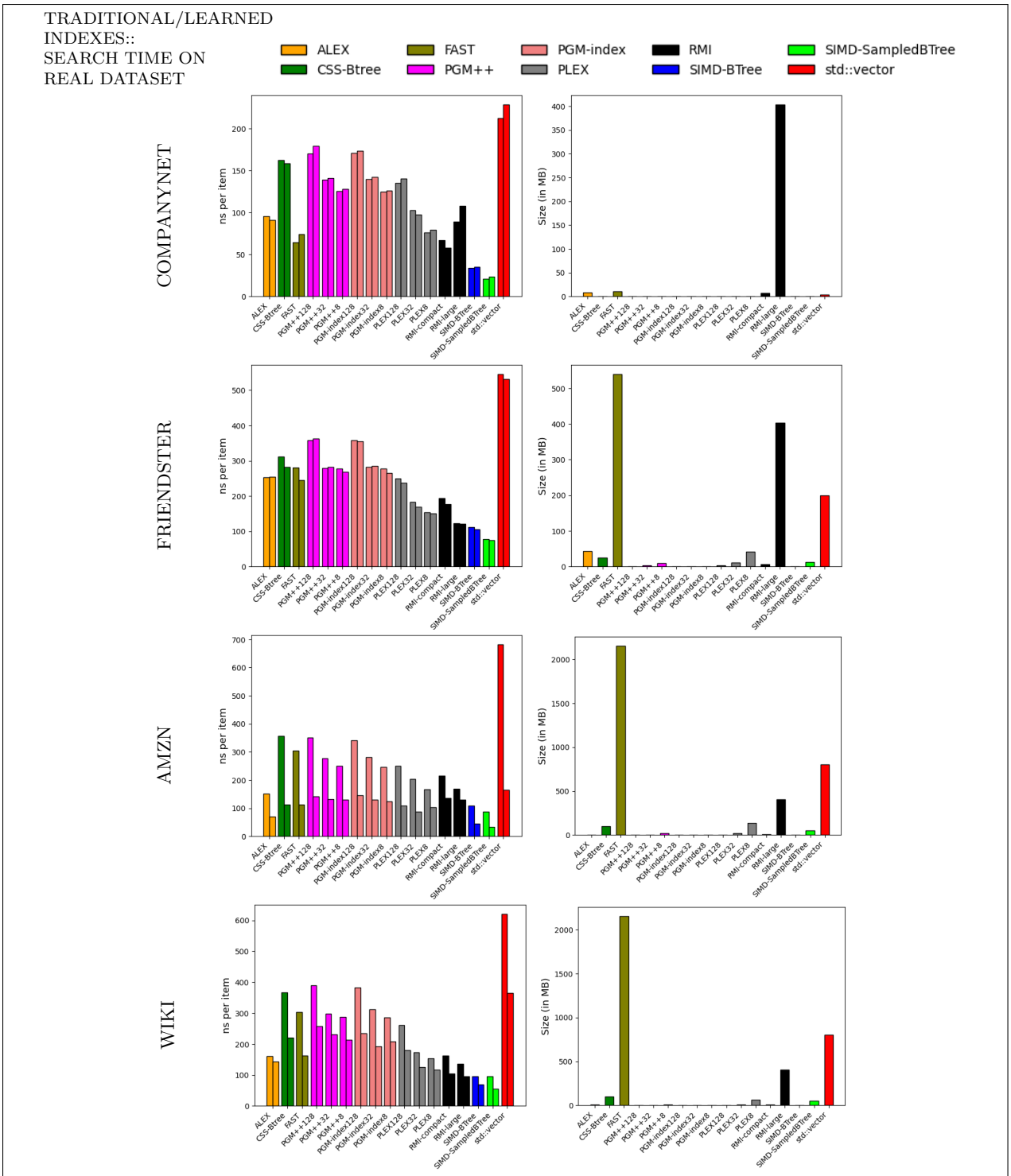
The plots above report the time required to build the data structures (indexes) on the synthetic datasets. On the left, the dataset's name is shown; in the middle, the complete plot with all the times expressed in milliseconds; and on the right, the same plot but excluding the ones with excessive build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

Figure 14: Build time on synthetic datasets



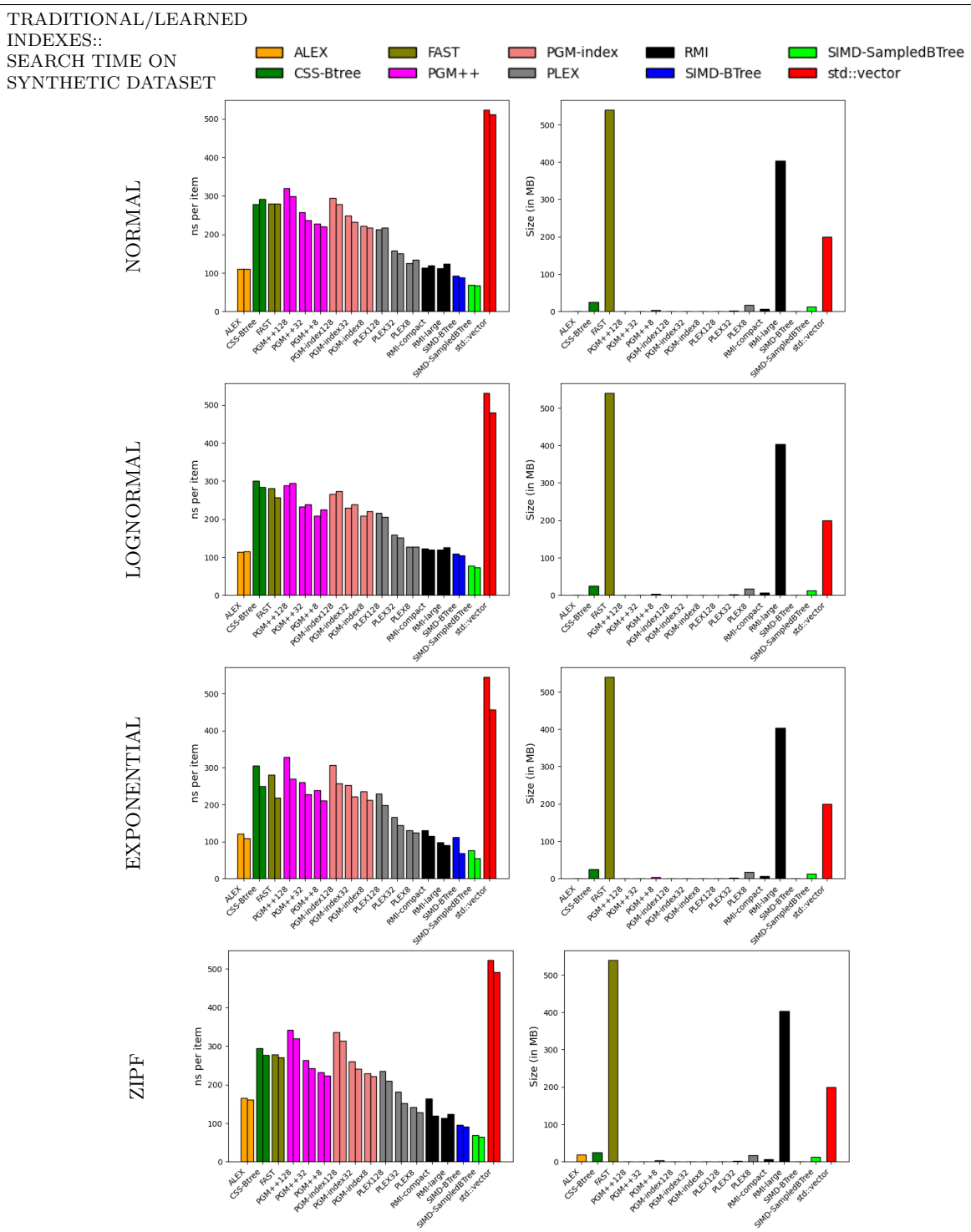
The plots above report the time required to build the data structures (indexes) on the 64-bit datasets. On the left, the dataset's name is shown; in the middle, the complete plot with all the times expressed in milliseconds; and on the right, the same plot but excluding the ones with excessive build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

Figure 15: Build time on 64-bit datasets



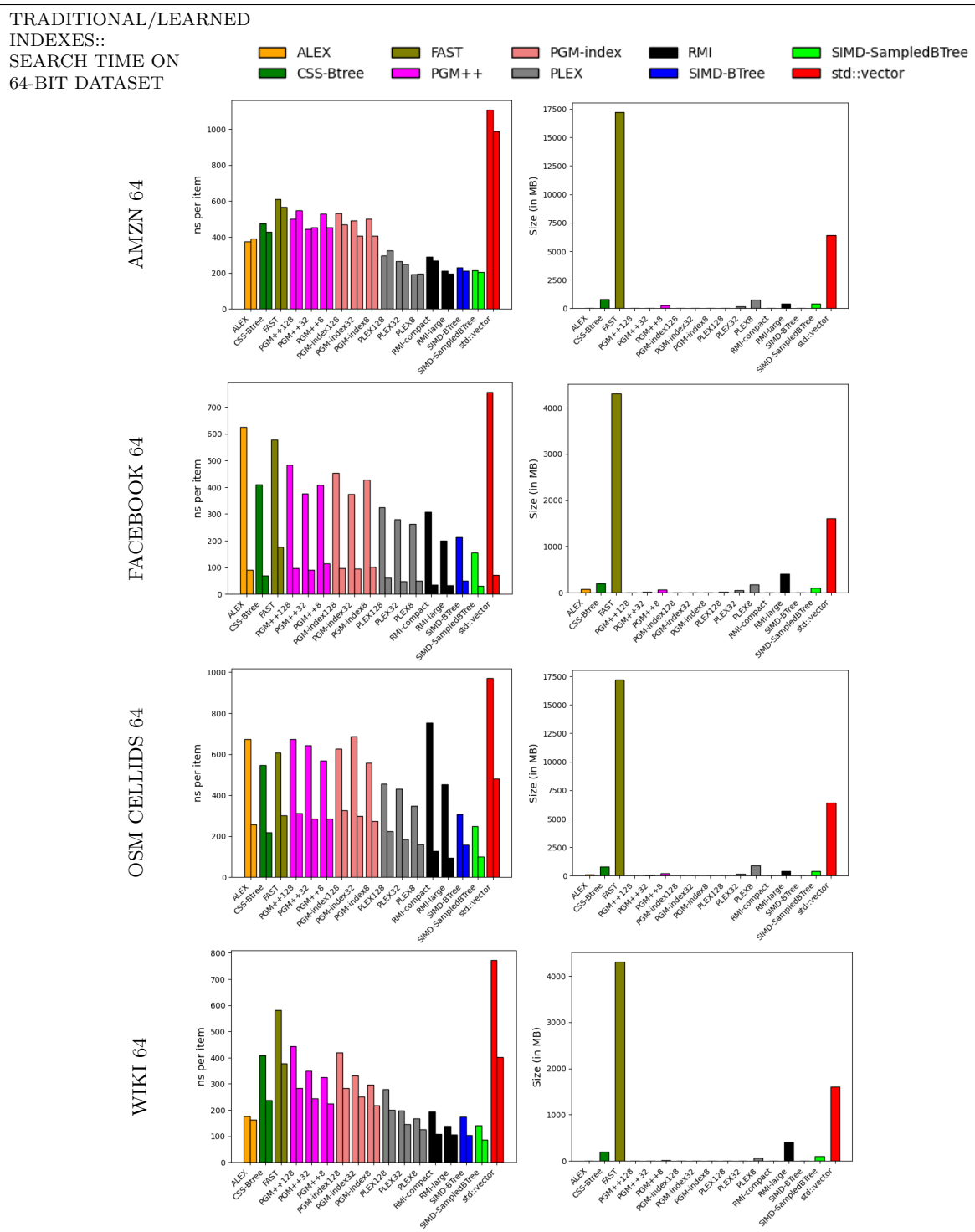
The performances of traditional and learned indexes on real-world datasets are represented in two plots. The leftmost plot shows the time of search (in ns) on each index; while the right plot shows the space occupied by each index in MBs (without considering the space occupied by the data to be stored). 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.



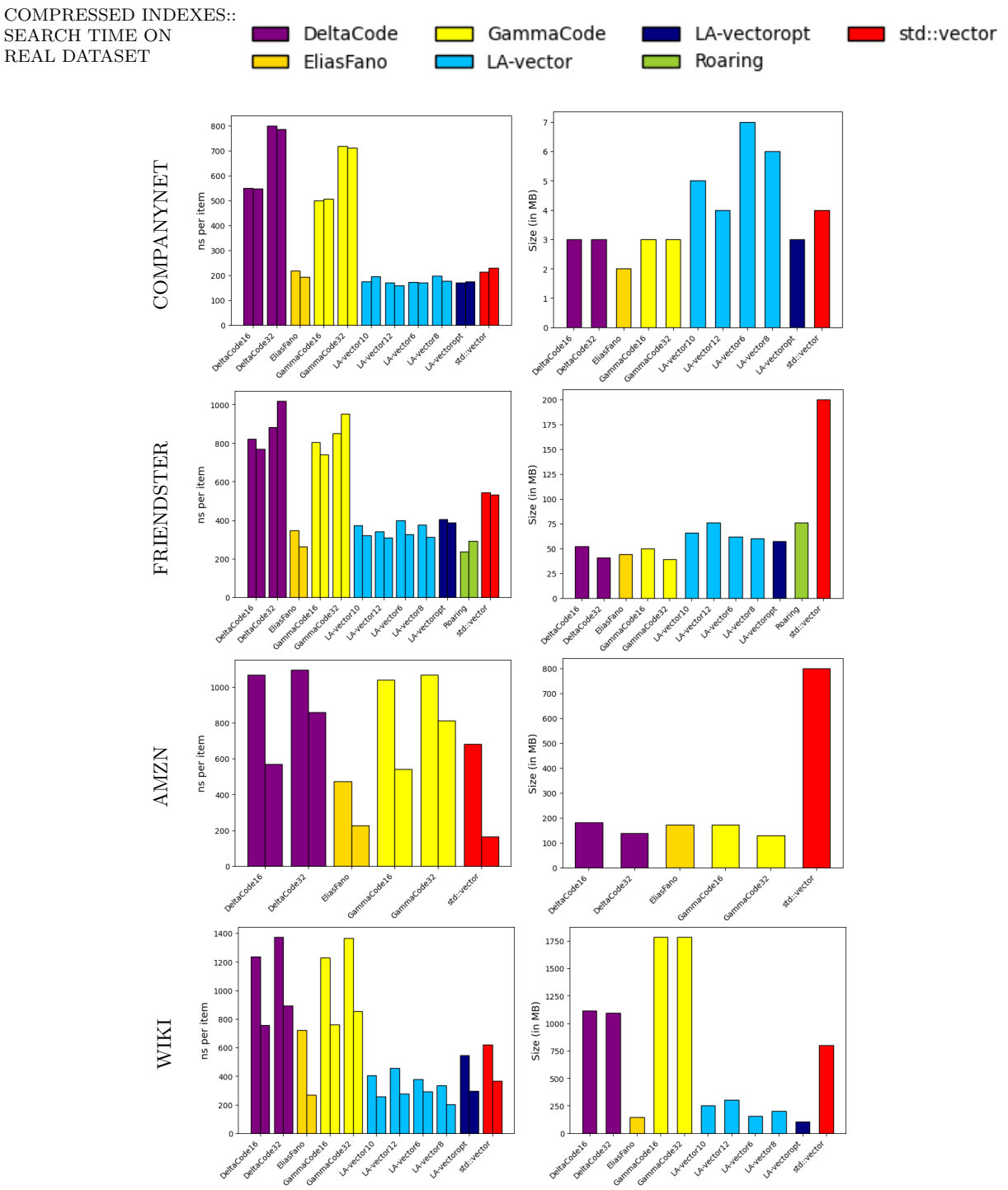
The performance of traditional and learned indexes on synthetic datasets are represented in two plots. The leftmost plot shows the time of search (in ns) on each index; while the right plot shows the space occupied by each index in MBs (without considering the space occupied by the data to be stored). 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.



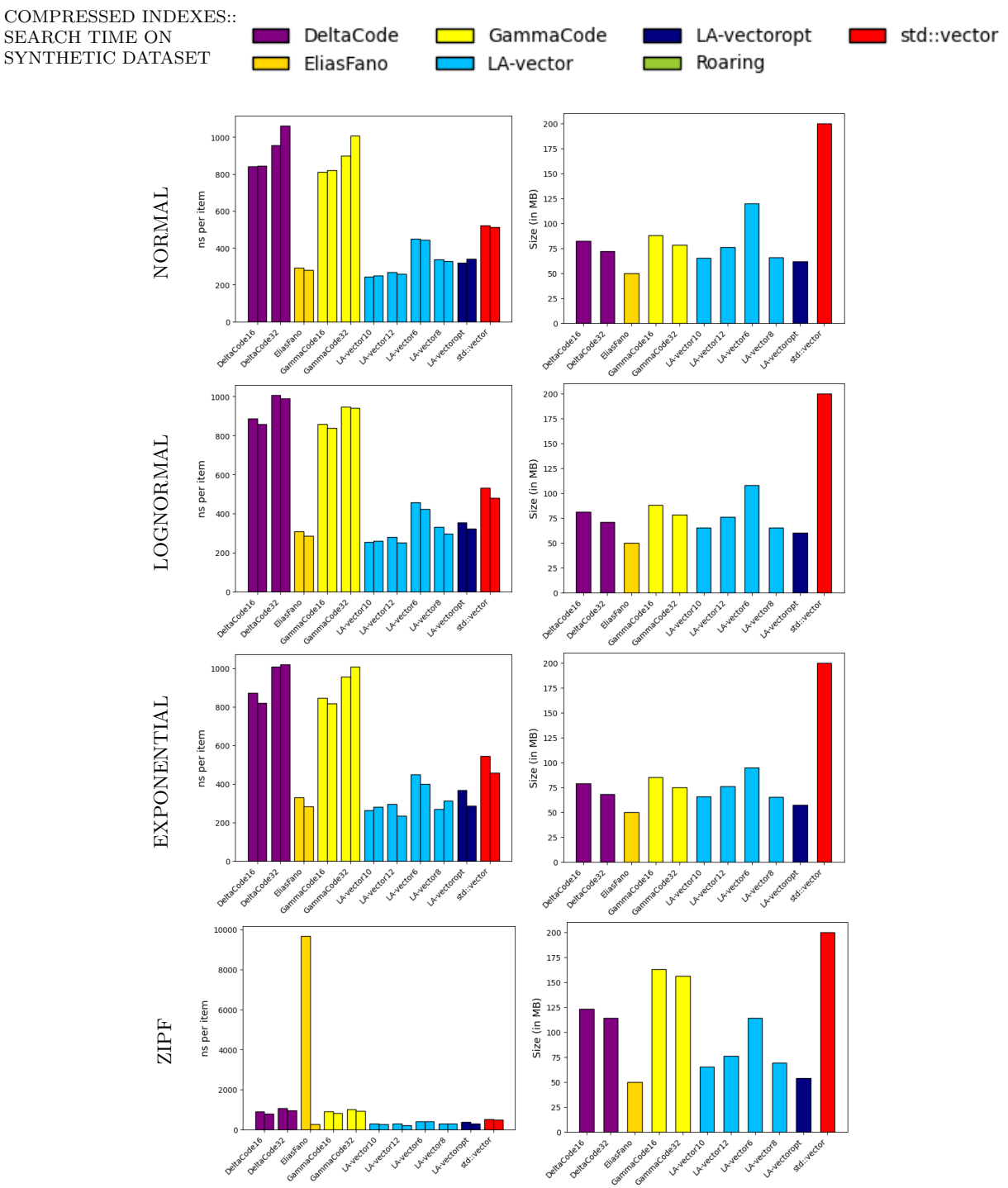
The performances of traditional and learned indexes on 64-bit datasets are represented in two plots. The leftmost plot shows the time of search (in ns) on each index; while the right plot shows the space occupied by each index in MBs (without considering the space occupied by the data to be stored). 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.



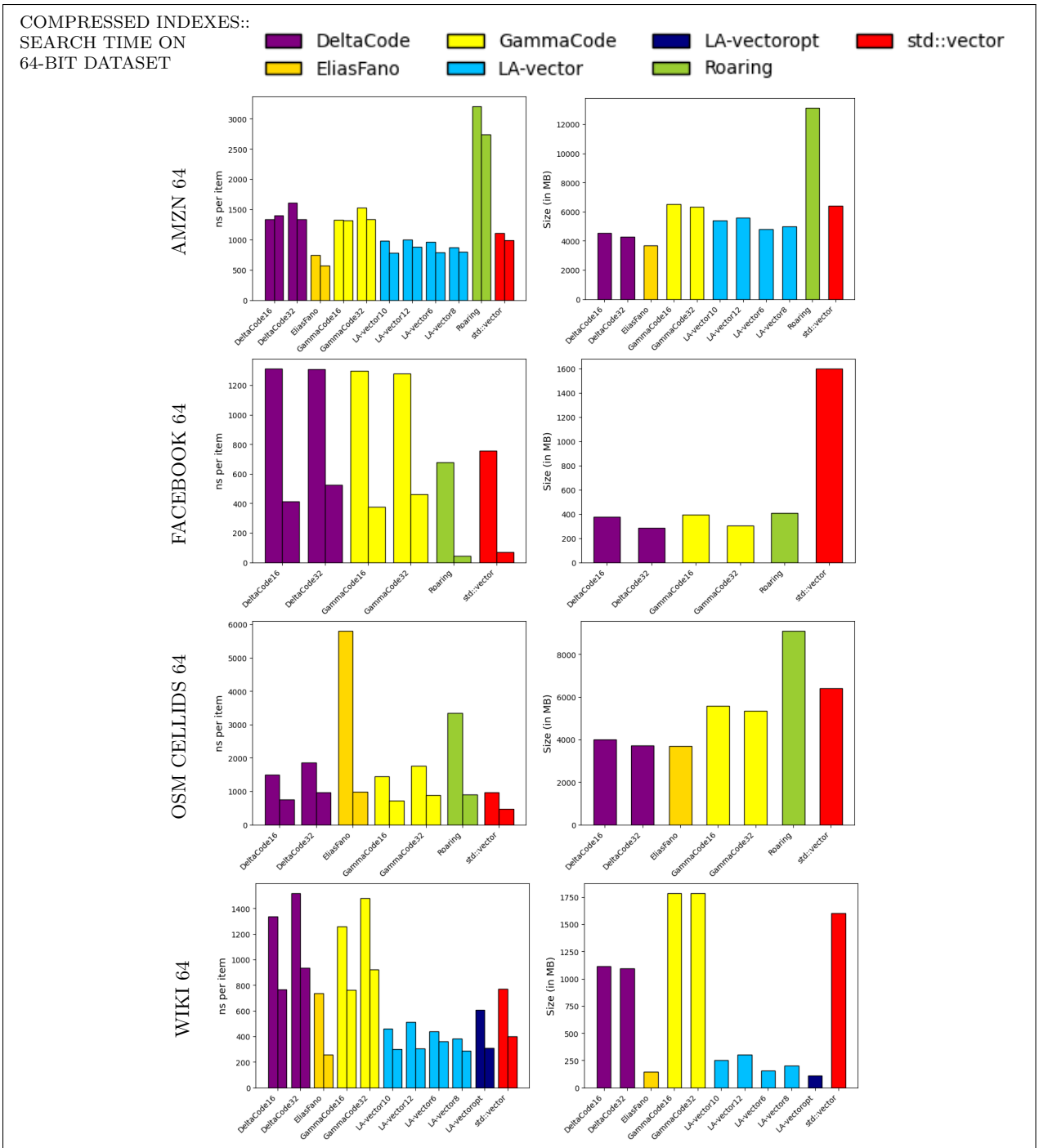
The performances of compressed indexes on real-world datasets are represented in two plots. The leftmost plot shows the time of search (in ns) on each index; while the right plot shows the space occupied by each index in MBs (without considering the space occupied by the data to be stored). The left 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.



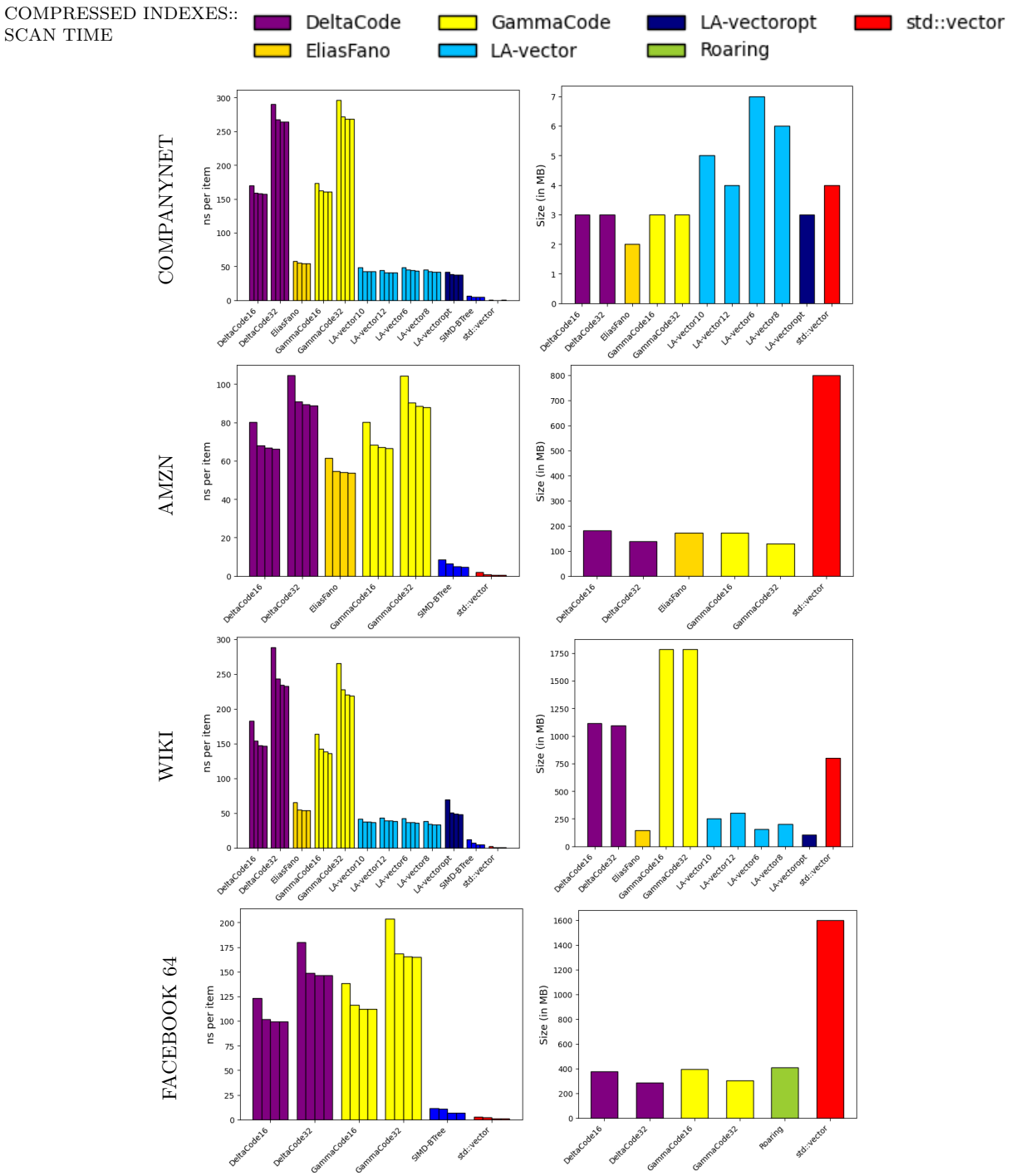
The performances of compressed indexes on synthetic datasets are represented in two plots. The leftmost plot shows the time of search (in ns) on each index; while the right plot shows the space occupied by each index in MBs (without considering the space occupied by the data to be stored). The left plot 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.



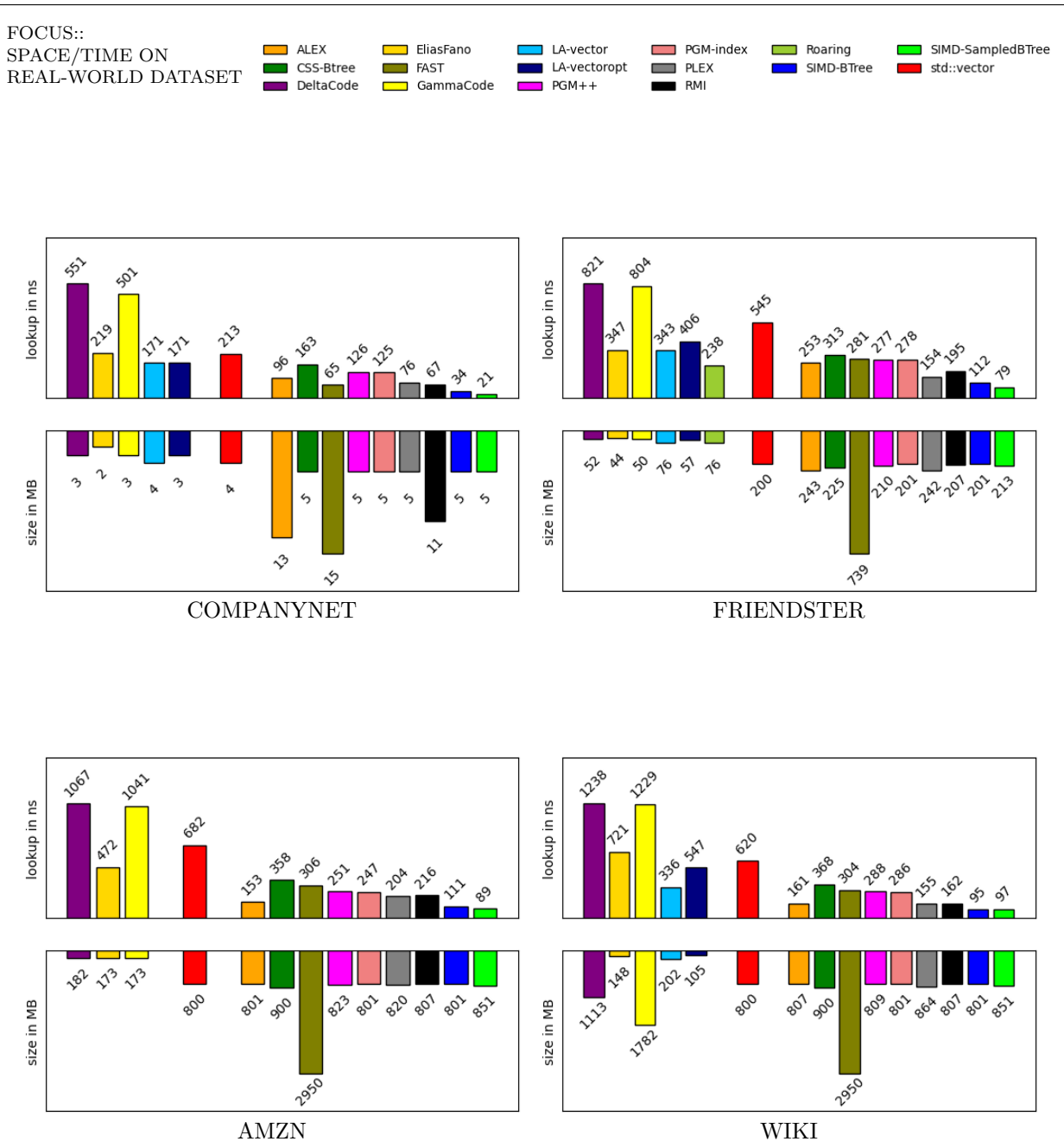
The performances of compressed indexes on 64-bit datasets are represented in two plots. The leftmost plot shows the time of search (in ns) on each index; while the right plot shows the space occupied by each index in MBs (without considering the space occupied by the data to be stored). The left plot 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.



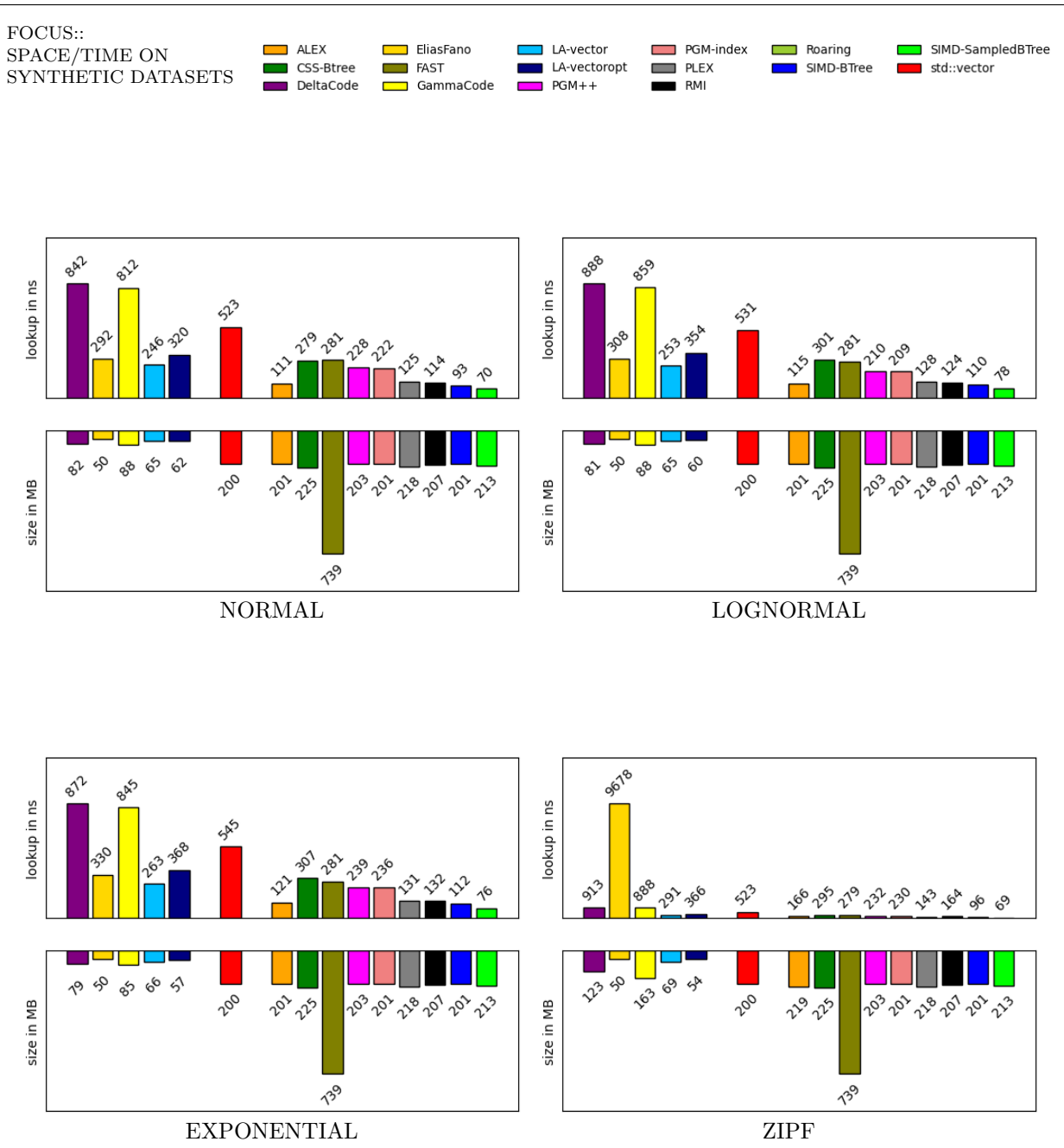
The plots above show the performance of compressed indexes in time and space relative to *range queries*, where starting points are randomly sampled, and the width of the scan is set to 10, 100, 1K, and 10K. The plot on the left shows the time (in ns) required for every interrogation, describing the 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.



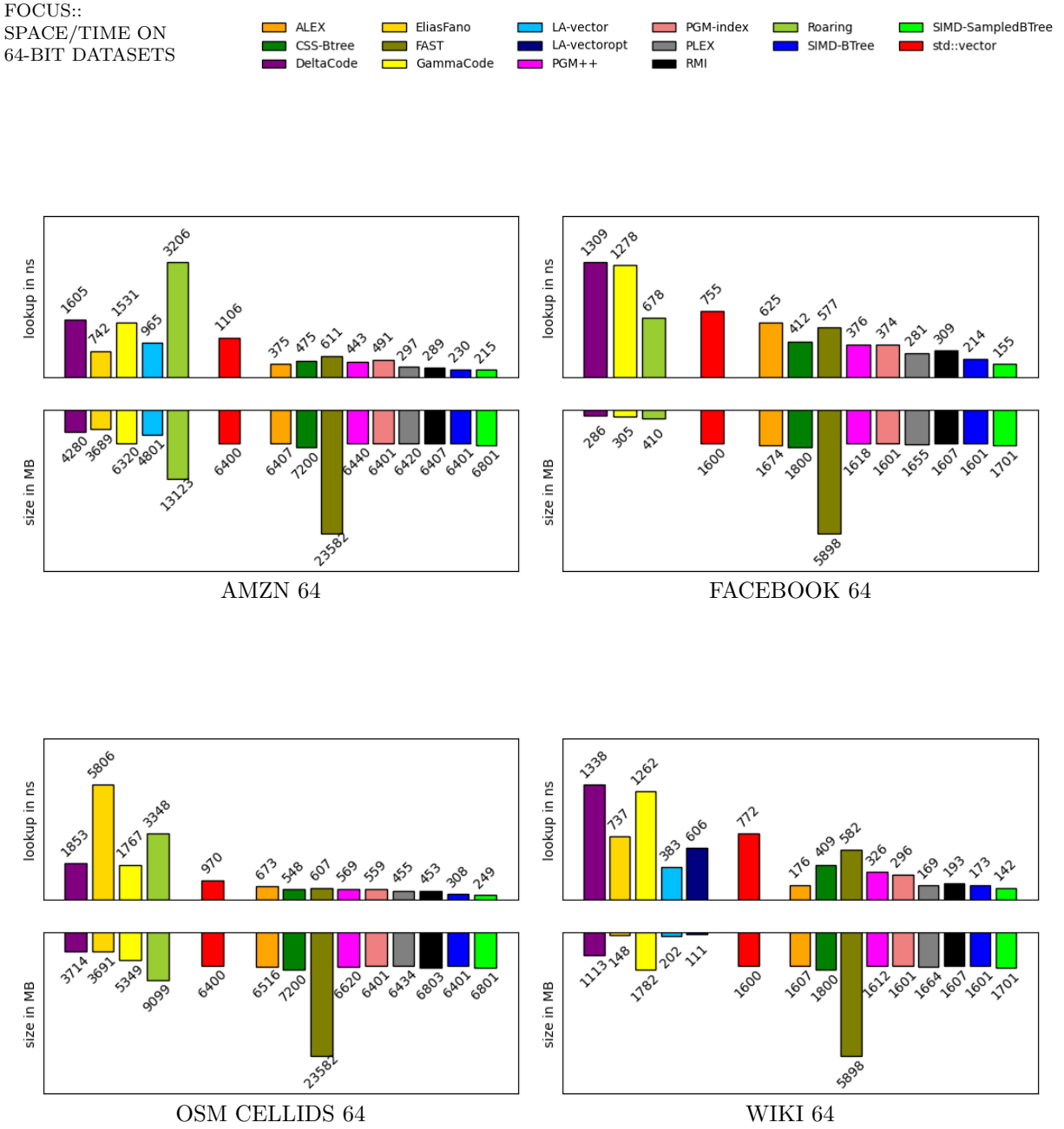
The figures above show the results corresponding to the space occupied and elapsed time for pointwise queries on real-world datasets on all tested indexes (traditional, learned, and compressed). For each dataset and for each index, the top part shows the average time (in ns) needed to make a query on 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.



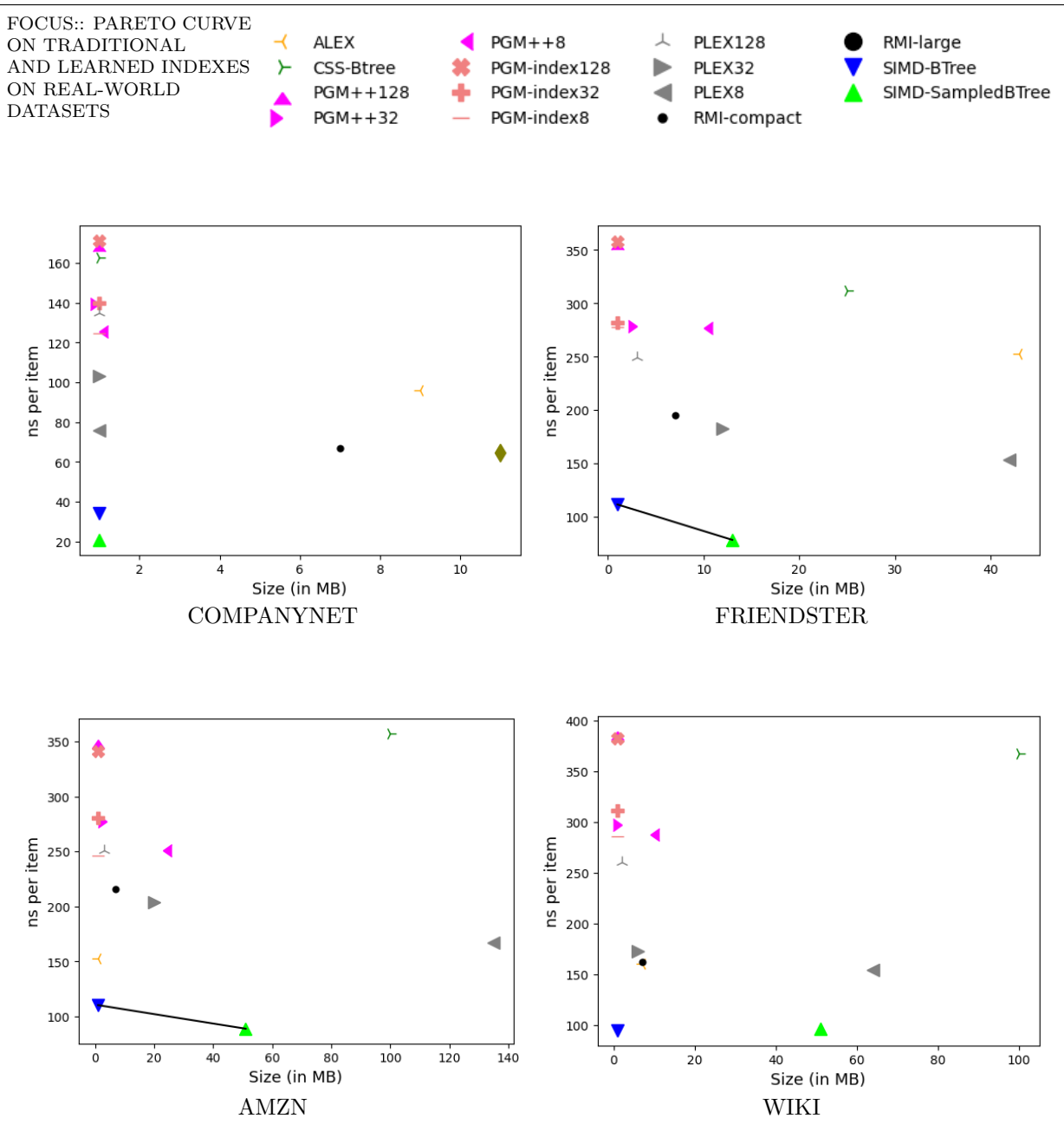
The figures above show the results corresponding to the space occupied and elapsed time for pointwise queries on synthetic datasets on all tested indexes (traditional, learned, and compressed). For each dataset and for each index, the top part shows the average time (in ns) needed to make a query on 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.

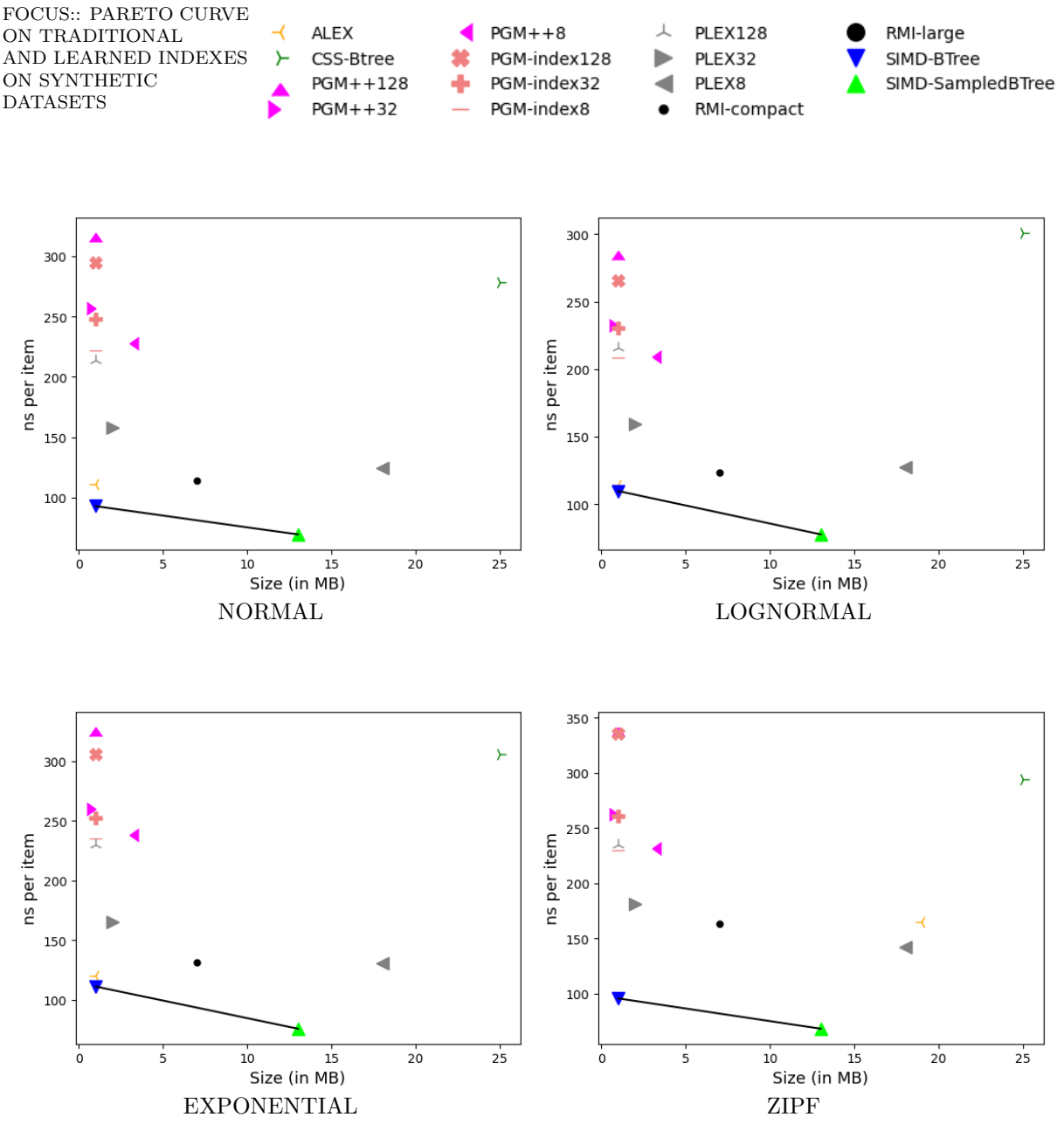
Figure 25: Recap space/time plots for pointwise queries on 64-bit datasets.



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.

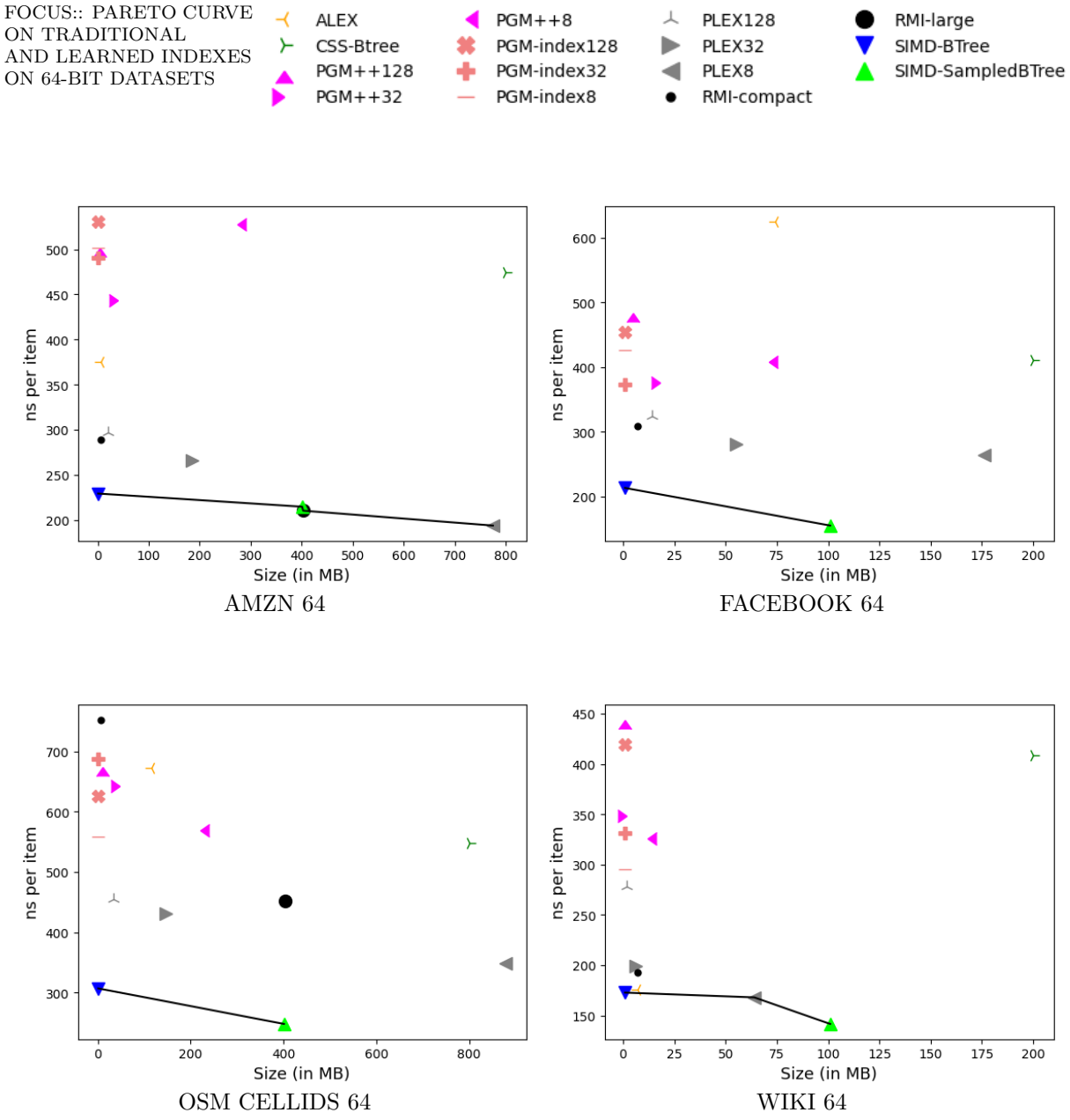
Figure 26: Pareto Frontier: Traditional and Learned Indexes on real-world datasets



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.

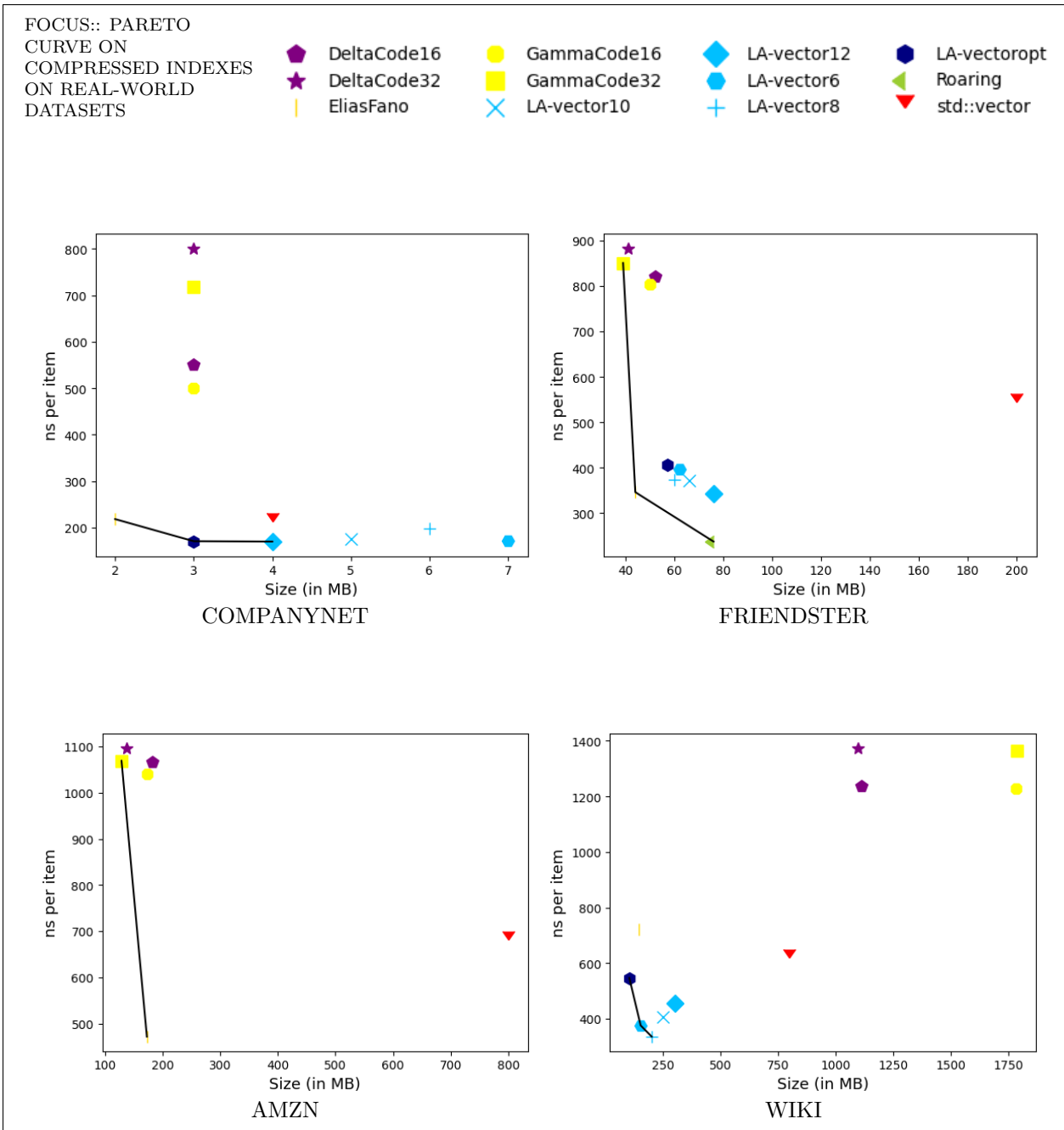
Figure 27: Pareto Frontier: Traditional and Learned Indexes on synthetic datasets



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.

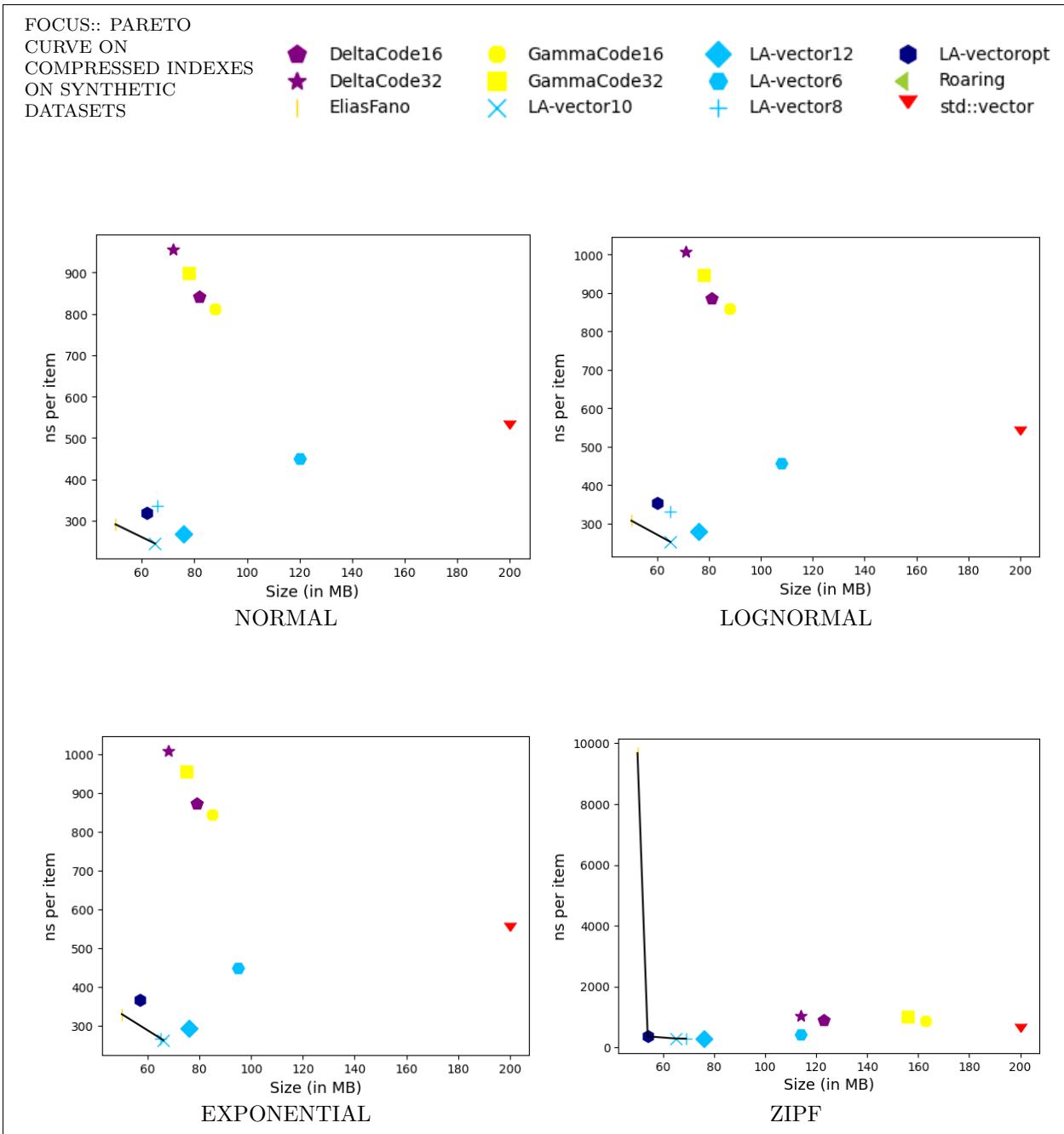
Figure 28: Pareto Frontier: Traditional and Learned Indexes on 64-bit datasets



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.

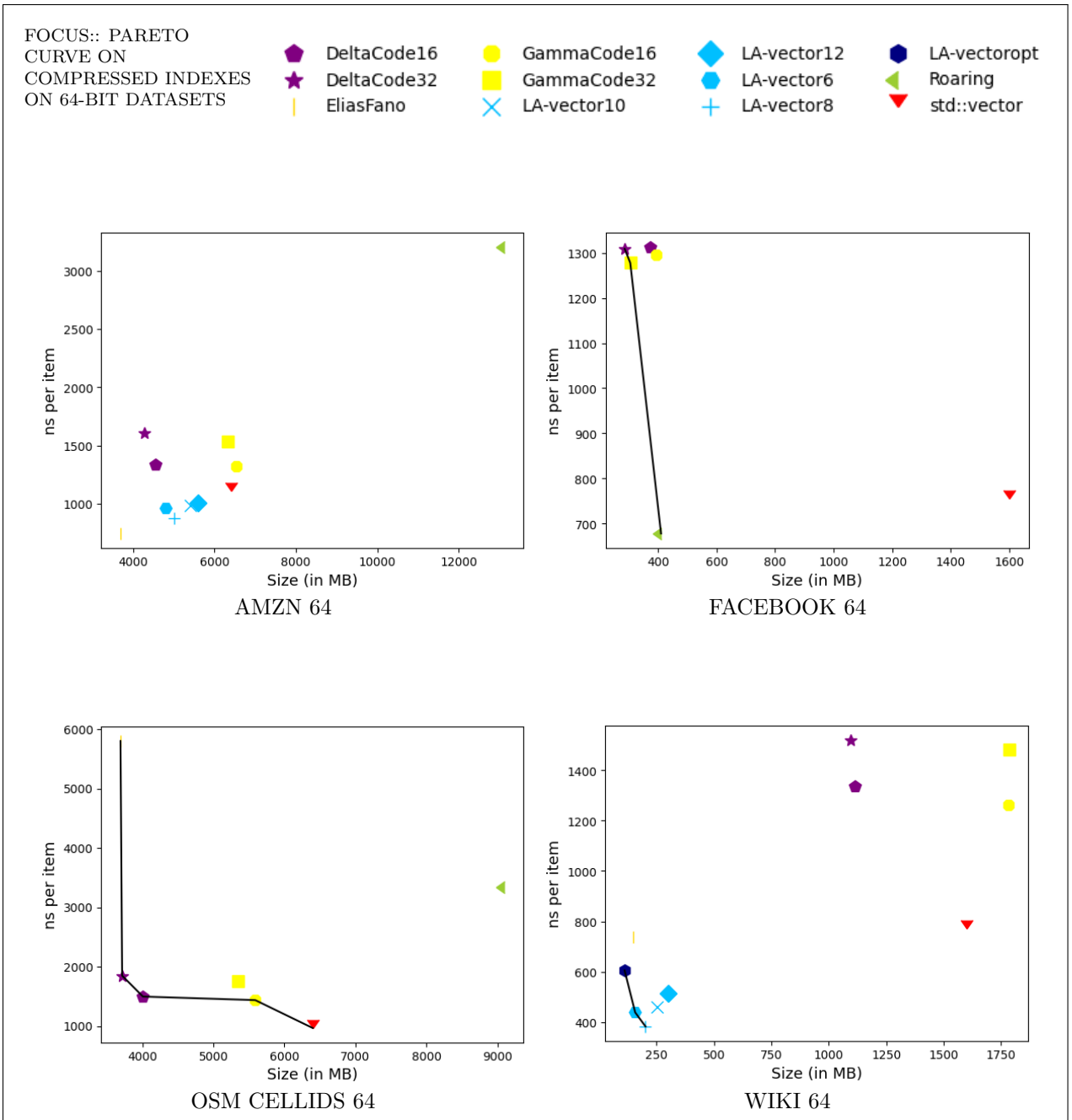
Figure 29: Pareto Frontier: Compressed Indexes on real-world datasets



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.

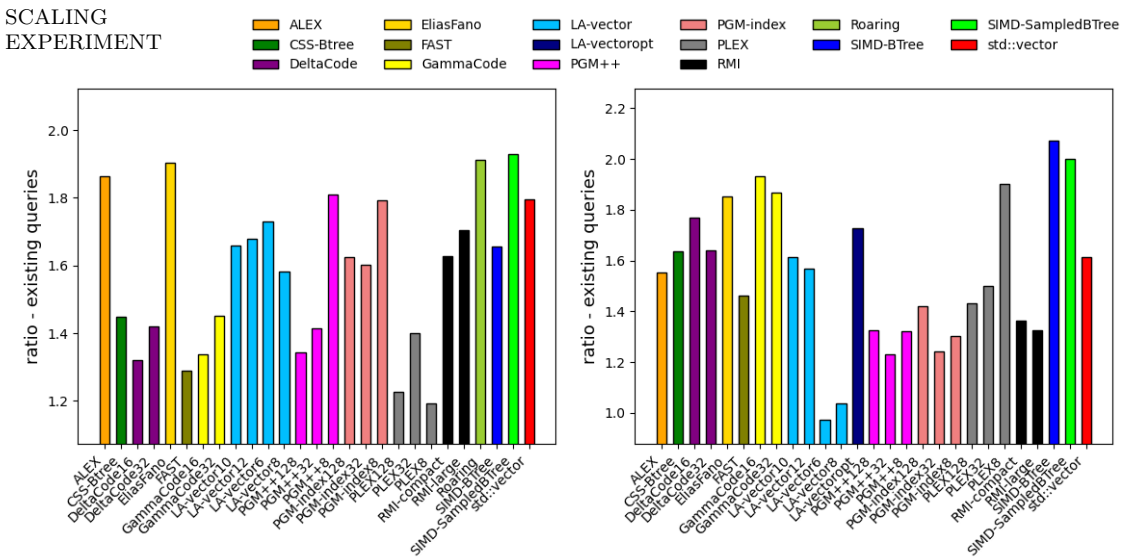
Figure 30: Pareto Frontier: Compressed Indexes on synthetic datasets



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



The performance of all indexes on the Amzn64 and Normal datasets is represented in two plots. The plot shows the ratio for a query of an existing item. In the case of Amzn, the ratio considers the performance of indexes on the 50M slice of Amzn64, and Amzn64 in its entirety. In the case of Normal, the ratio corresponds to the performance of indexes on the 50M syntetically generated Normal dataset, and to another Normal dataset, generated using the same parameters, but of size 800M. In both cases, the performance refers to the ratio of the average ns needed to find the item in the large dataset, divided by the average ns needed to find an item in the smaller scale dataset.

Figure 32: Average time ratio for pointwise queries on all indexes when scaling the dataset size (50M to 800M items).

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	95.9091	90.9808	9	0.0676480358466506
CSS-Btree	162.82	158.742	1	0.0031119640916585922
DeltaCode16	550.807	546.701	3	0.02400585152208805
DeltaCode32	800.797	787.052	3	0.02441169861704111
EliasFano	218.418	191.767	2	0.03514758311212063
FAST	64.5389	74.1089	11	0.010569458082318305
GammaCode16	500.809	507.477	3	0.018579979427158833
GammaCode32	717.836	711.566	3	0.018837350606918334
LA-vector10	175.483	194.521	5	0.0453252611681819
LA-vector12	170.13	158.736	4	0.03015019092708826
LA-vector6	171.538	170.396	7	0.05389650873839855
LA-vector8	197.474	177.544	6	0.04818133469671011
LA-vectoropt	170.858	174.579	3	0.850500600039959
PGM++128	170.362	179.161	1	0.002402110770344734
PGM++32	139.233	141.255	1	0.002361966110765934
PGM++8	125.744	127.946	1	0.0033521194010972975
PGM-index128	171.263	173.753	-	0.002394743077456951
PGM-index32	139.919	142.348	-	0.0023480268195271493
PGM-index8	124.813	126.459	-	0.003352062217891216
PLEX128	135.145	140.614	1	0.01870745625346899
PLEX32	103.076	97.3781	1	0.020280578173696995
PLEX8	75.9571	79.241	1	0.02445698659867048
RMI-compact	66.9319	58.0909	7	0.0010586014017462731
RMI-large	89.2875	108.191	403	0.11085550598800183
SIMD-BTree	33.9081	35.5578	1	0.003077902086079121
SIMD-SampledBTree	20.7226	23.3043	1	0.0022979401051998138
std::vector	212.374	228.589	4	-

Table 1: Tabular data: companynet dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	160.457	143.458	7	12.012247908860445
CSS-Btree	367.274	220.321	100	1.28394063282758
DeltaCode16	1237.44	757.516	1113	7.681204113550484
DeltaCode32	1373.87	894.55	1095	7.646052028052508
EliasFano	720.522	269.429	148	7.920321106351912
FAST	303.357	163.379	2150	2.563926495797932
GammaCode16	1228.98	758.484	1782	7.785817414708435
GammaCode32	1365.74	852.905	1785	7.690992759726941
LA-vector10	406.119	257.723	251	1.643485777825117
LA-vector12	456.304	275.901	301	1.7391836095601318
LA-vector6	375.75	292.294	154	2.159774744324386
LA-vector8	335.261	201.141	202	1.7653099594637751
LA-vectoropt	546.652	297.504	105	189.3272948315367
PGM++128	389.503	258.515	1	0.4941654935479164
PGM++32	297.715	230.945	2	0.49058906733989716
PGM++8	287.676	214.33	9	0.5613158477470279
PGM-index128	382.776	234.716	-	0.4924962343648076
PGM-index32	311.588	192.884	-	0.4889260321855545
PGM-index8	285.538	209.002	-	0.5581602398306131
PLEX128	260.497	179.429	2	3.1324711384251716
PLEX32	172.882	125.639	6	3.3536606315523385
PLEX8	154.179	116.693	64	3.9040234006941312
RMI-compact	161.929	104.131	7	0.0009129747748374939
RMI-large	137.001	95.1037	403	0.10955970585346222
SIMD-BTree	94.8834	68.9426	1	0.9346063748002053
SIMD-SampledBTree	96.5711	55.5205	51	1.0938821226358413
std::vector	619.809	365.667	800	-

Table 2: Tabular data: wiki uint32 dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	252.526	254.674	43	3.240652223303914
CSS-Btree	312.144	281.64	25	0.19232455547899008
DeltaCode16	820.541	768.513	52	1.199598570726812
DeltaCode32	882.524	1018.65	41	1.2158852193504572
EliasFano	346.337	263.288	44	1.8770567055791618
FAST	280.278	244.701	539	0.6358619391918182
GammaCode16	803.233	740.176	50	0.9702335396781563
GammaCode32	850.444	950.521	39	0.9723387679085137
LA-vector10	371.448	320.223	66	0.44665191881358624
LA-vector12	342.136	308.392	76	0.4266238516196609
LA-vector6	397.683	327.079	62	0.8292287347838283
LA-vector8	374.293	312.799	60	0.5508538806810975
LA-vectoropt	405.477	386.56	57	48.53176510110497
PGM++128	358.675	363.241	1	0.13278031013906003
PGM++32	278.542	282.243	3	0.1610252683982253
PGM++8	276.863	267.694	10	0.20582125056535006
PGM-index128	357.917	354.844	-	0.1329347249120474
PGM-index32	281.758	285.042	-	0.16006165463477373
PGM-index8	277.706	265.747	-	0.20458792969584466
PLEX128	249.423	237.454	3	0.9966949267312886
PLEX32	182.589	169.652	12	1.1849591376259923
PLEX8	153.059	150.775	42	1.4839747536927461
RMI-compact	194.762	177.412	7	0.0009729171171784401
RMI-large	122.766	121.169	403	0.10234209969639778
Roaring	237.651	292.729	76	2.1193305596709253
SIMD-BTree	111.46	105.082	1	0.17117644492536782
SIMD-SampledBTree	78.3161	74.9588	13	0.15019659120589496
std::vector	544.724	531.255	200	-

Table 3: Tabular data: friendster dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	120.489	108.177	1	1.7942154103890062
CSS-Btree	306.014	248.896	25	0.20147268623113632
DeltaCode16	871.67	817.712	79	1.2446506787091494
DeltaCode32	1008.37	1019.77	68	1.2687088703736662
EliasFano	329.865	284.505	50	1.5662505311891437
FAST	280.336	218.442	539	0.6370683642104268
GammaCode16	844.335	816.176	85	0.9665239047259092
GammaCode32	953.931	1008.78	75	0.9840111192315817
LA-vector10	262.36	280.22	66	0.46774770971387625
LA-vector12	293.974	235.206	76	0.4233253363519907
LA-vector6	447.991	397.737	95	1.3631610522046684
LA-vector8	267.271	313.109	65	0.6580481169745326
LA-vectoropt	367.144	285.315	57	45.88938278835267
PGM++128	328.592	269.896	1	0.08260615151375532
PGM++32	260.059	228.504	1	0.09546069372445345
PGM++8	238.567	211.365	3	0.13138940203934907
PGM-index128	306.314	257.338	-	0.08251420743763446
PGM-index32	252.913	222.319	-	0.09519395157694817
PGM-index8	235.092	212.287	-	0.13072128295898439
PLEX128	229.806	197.78	1	0.906862278096378
PLEX32	165.319	144.097	2	1.0425597973167897
PLEX8	130.64	124.45	18	1.2915953338146209
RMI-compact	131.257	115.707	7	0.0009223395958542824
RMI-large	98.6948	90.4686	403	0.11102965399622917
SIMD-BTree	111.264	68.5444	1	0.17103358041495084
SIMD-SampledBTree	75.9218	54.8878	13	0.1500023901462555
std::vector	544.363	457.192	200	-

Table 4: Tabular data: exponential dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	114.336	114.611	1	1.8290619304403664
CSS-Btree	300.901	283.294	25	0.19371440429240466
DeltaCode16	887.255	856.913	81	1.2674335973337292
DeltaCode32	1007.89	990.618	71	1.2753148147836328
EliasFano	307.804	285.517	50	1.6140515057370066
FAST	280.295	257.103	539	0.6595098756253719
GammaCode16	858.566	839.114	88	0.9756283687427639
GammaCode32	947.235	939.92	78	0.9861554896458985
LA-vector10	252.852	260.096	65	0.44184423983097076
LA-vector12	278.684	251.678	76	0.4187323458492756
LA-vector6	458.247	423.295	108	1.3490668019279837
LA-vector8	332.342	296.581	65	0.6395080419257283
LA-vectoropt	353.554	322.54	60	46.33683096393943
PGM++128	287.758	294.664	1	0.08448264207690954
PGM++32	232.615	238.919	1	0.09548578225076199
PGM++8	209.089	225.152	3	0.13308370690792798
PGM-index128	265.443	273.856	-	0.08426300901919603
PGM-index32	230.147	238.935	-	0.09486935958266259
PGM-index8	208.162	221.011	-	0.13067508675158024
PLEX128	215.927	206.187	1	0.9050493353977799
PLEX32	159.216	150.827	2	1.0416429514065384
PLEX8	127.124	127.154	18	1.306325313448906
RMI-compact	123.256	119.397	7	0.0009855557233095168
RMI-large	119.676	125.852	403	0.11240628454834223
SIMD-BTree	109.688	104.008	1	0.17056054193526507
SIMD-SampledBTree	77.5731	72.6102	13	0.15220521669834852
std::vector	530.59	479.343	200	-

Table 5: Tabular data: lognormal dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	165.137	161.121	19	3.2173378605395557
CSS-Btree	294.004	276.042	25	0.19199986718595027
DeltaCode16	912.961	801.816	123	1.3220081066712737
DeltaCode32	1054.81	953.704	114	1.3333910105749964
EliasFano	9677.49	277.445	50	1.6184972804039717
FAST	278.253	269.922	539	0.6379272405058145
GammaCode16	887.639	819.475	163	1.0873271148651837
GammaCode32	1003.51	930.293	156	1.088339378684759
LA-vector10	293.5	267.501	65	0.4485928285866976
LA-vector12	299.915	220.828	76	0.40411267913877963
LA-vector6	415.163	405.647	114	1.479838857613504
LA-vector8	290.116	295.279	69	0.668976902961731
LA-vectoropt	365.163	308.81	54	45.89293846506625
PGM++128	341.446	319.4	1	0.08525088038295507
PGM++32	262.52	242.604	1	0.09717893153429032
PGM++8	231.426	223.748	3	0.13262233808636664
PGM-index128	335.569	312.844	-	0.08477807193994522
PGM-index32	260.479	241.404	-	0.09708657581359148
PGM-index8	229.575	221.6	-	0.13302721306681634
PLEX128	234.982	209.51	1	0.8541710514575243
PLEX32	181.431	151.634	2	0.9811131335794926
PLEX8	142.146	127.712	18	1.2192370543256401
RMI-compact	163.146	118.945	7	0.0008021021261811256
RMI-large	113.424	123.584	403	0.10623279083520173
SIMD-BTree	95.6123	90.8948	1	0.16785447504371404
SIMD-SampledBTree	68.1713	63.9113	13	0.15028309132903814
std::vector	522.126	492.004	200	-

Table 6: Tabular data: zipf dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	110.621	110.553	1	1.5848807588219642
CSS-Btree	278.192	291.335	25	0.1917602563276887
DeltaCode16	841.201	845.545	82	1.2471350852400065
DeltaCode32	956.187	1062.07	72	1.2736568097025156
EliasFano	291.339	280.35	50	1.5886168884113432
FAST	280.264	278.988	539	0.6478388475254178
GammaCode16	811.234	821.372	88	0.9679135221987962
GammaCode32	897.956	1005.59	78	0.9798693276941777
LA-vector10	245.105	248.752	65	0.4484759349375963
LA-vector12	268.748	257.887	76	0.4031166210770607
LA-vector6	450.018	442.77	120	1.522797609679401
LA-vector8	336.793	329.26	66	0.6341059608384967
LA-vectoropt	319.987	341.684	62	46.16744492799043
PGM++128	319.447	299.125	1	0.08144631292670965
PGM++32	256.676	236.917	1	0.09477787278592587
PGM++8	227.5	220.876	3	0.13118064105510713
PGM-index128	294.702	277.936	-	0.08104249183088541
PGM-index32	248.029	232.884	-	0.09447404090315104
PGM-index8	221.851	217.76	-	0.13248117361217737
PLEX128	213.339	217.549	1	0.9024829858914016
PLEX32	157.824	151.171	2	1.0398293994367123
PLEX8	124.688	133.787	18	1.3045564578846096
RMI-compact	113.593	120.059	7	0.0008146867156028747
RMI-large	111.776	123.063	403	0.10977681111544371
SIMD-BTree	92.7386	88.5321	1	0.17308539804071188
SIMD-SampledBTree	69.3477	67.2098	13	0.15776039082556964
std::vector	522.569	511.753	200	-

Table 7: Tabular data: normal dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	152.191	69.8671	1	83.61921926215291
CSS-Btree	357.266	112.172	100	0.7957991672679782
DeltaCode16	1066.97	568.682	182	5.29572212714702
DeltaCode32	1096.46	857.466	138	4.943631635420024
EliasFano	471.399	225.247	173	8.63053139206022
FAST	305.239	112.682	2150	4.470826370641589
GammaCode16	1040.27	540.943	173	3.902928456105292
GammaCode32	1068.95	813.301	129	3.9591487292200327
PGM++128	351.538	141.269	1	0.4213562335819006
PGM++32	276.924	132.289	4	0.5384868007153273
PGM++8	250.746	130.292	23	0.712211299687624
PGM-index128	341.25	146.594	-	0.4173959335312247
PGM-index32	280.67	129.823	-	0.5418468924239278
PGM-index8	246.209	125.135	-	0.7263280685991049
PLEX128	251.034	109.065	3	3.349527246132493
PLEX32	203.742	87.336	20	3.711609287559986
PLEX8	166.568	103.643	135	4.838791682757437
RMI-compact	215.751	136.266	7	0.0016414130106568337
RMI-large	170.075	129.677	403	0.13251978754997254
SIMD-BTree	110.105	44.6701	1	0.7171605022624135
SIMD-SampledBTree	88.6804	34.3279	51	0.6088854754343629
std::vector	681.43	165.667	800	-

Table 8: Tabular data: amzn uint32 dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	374.923	389.146	7	51.52129691876471
CSS-Btree	474.201	429.052	800	14.276615456864237
DeltaCode16	1335.58	1396.3	4554	31.874918396957224
DeltaCode32	1604.87	1332.01	4280	31.915880217403174
EliasFano	741.849	574.616	3689	36.76156809199602
FAST	610.17	567.577	17182	20.307497927173973
GammaCode16	1323.89	1321.93	6529	28.784663809649647
GammaCode32	1530.06	1337.57	6320	28.60775261074305
LA-vector10	984.989	781.965	5401	93.86812636367976
LA-vector12	1001.86	877.305	5601	103.12081321552395
LA-vector6	964.467	792.334	4801	93.34385407399387
LA-vector8	872.641	795.422	5001	92.35115467403084
PGM++128	501.056	548.715	5	2.0942088013514875
PGM++32	442.952	454.883	40	2.566647768206894
PGM++8	527.651	454.074	273	3.749479307793081
PGM-index128	530.668	467.596	-	2.0387150306254624
PGM-index32	490.177	405.586	-	2.593148315139115
PGM-index8	501.892	406.031	-	3.7587563183158634
PLEX128	296.687	323.45	20	14.835273451358082
PLEX32	265.755	247.993	186	17.064765594527124
PLEX8	193.927	195.778	775	22.902524994313715
RMI-compact	288.912	269.172	7	0.0010563431307673455
RMI-large	210.4	194.854	403	0.10752807408571244
Roaring	3205.13	2742.86	13123	266.1944434076548
SIMD-BTree	229.39	211.936	1	11.644063972495495
SIMD-SampledBTree	214.785	206.259	401	14.291339065134526
std::vector	1105.36	987.029	6400	-

Table 9: Tabular data: amzn uint64 dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	672.733	257.394	116	600.1624881919474
CSS-Btree	547.535	219.761	800	11.340442677587271
DeltaCode16	1500.42	746.898	4007	26.093982944265008
DeltaCode32	1852.75	964.362	3714	26.196357237547634
EliasFano	5805.71	992.237	3691	40.82788153197617
FAST	606.79	301.625	17182	17.76152748707682
GammaCode16	1440.56	714.11	5589	24.477984689734875
GammaCode32	1766.32	885.395	5349	24.4328909477219
PGM++128	673.986	312.729	11	2.443368702381849
PGM++32	642.252	285.794	48	2.743877573125064
PGM++8	568.408	285.756	220	3.810089155286551
PGM-index128	625.91	325.311	-	2.4578521784394978
PGM-index32	687.031	299.792	-	2.736572047136724
PGM-index8	558.378	273.957	-	3.8246812472119927
PLEX128	454.951	223.332	34	17.062723525986076
PLEX32	430.784	184.987	147	19.04731353595853
PLEX8	349.158	160.647	879	25.424416487663983
RMI-compact	752.173	127.952	7	0.0019135408103466034
RMI-large	452.468	94.559	403	0.3077841537073255
Roaring	3347.41	901.296	9099	185.59363163653762
SIMD-BTree	307.204	157.146	1	9.821308633685112
SIMD-SampledBTree	248.356	100.825	401	11.521748039498926
std::vector	969.653	479.204	6400	-

Table 10: Tabular data: OSM cellids dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	175.737	162.829	7	12.615637925267219
CSS-Btree	408.578	236.886	200	2.921221328154206
DeltaCode16	1337.46	766.214	1113	8.120892321504652
DeltaCode32	1519.41	935.943	1095	8.0516182359308
EliasFano	736.498	259.594	148	7.866881237179041
FAST	581.1	378.234	4298	4.127908423170448
GammaCode16	1261.15	764.337	1782	8.79435874838382
GammaCode32	1480.6	924.672	1785	8.735569733195007
LA-vector10	460.317	298.93	251	3.470297888852656
LA-vector12	513.41	304.942	301	3.4853176360949876
LA-vector6	439.035	361.964	157	3.414507610164583
LA-vector8	382.906	289.148	202	3.430257202684879
LA-vectoropt	605.474	311.443	111	248.7509241387248
PGM++128	443.707	284.325	1	0.5435256343334913
PGM++32	348.546	245.009	2	0.5437586519867181
PGM++8	325.783	223.799	12	0.6213311728090047
PGM-index128	419.792	283.732	-	0.5268160339444876
PGM-index32	331.193	250.675	-	0.5262913007289172
PGM-index8	295.632	218.299	-	0.6065858380869031
PLEX128	278.41	201.041	2	3.200195953808725
PLEX32	198.957	144.642	6	3.321335105411708
PLEX8	168.104	125.225	64	4.034811076894402
RMI-compact	192.988	107.7	7	0.001353432983160019
RMI-large	138.794	105.73	403	0.30923905968666077
SIMD-BTree	172.884	103.79	1	2.5429605431854725
SIMD-SampledBTree	141.763	85.9377	101	2.86786035541445
std::vector	771.45	402.192	1600	-

Table 11: Tabular data: wiki uint64 dataset

index	lookup existing (ns)	lookup missing (ns)	space (MB)	build (s)
ALEX	624.991	89.9834	74	17.097580939903853
CSS-Btree	411.324	70.4466	200	1.3400057671591639
DeltaCode16	1312.73	413.469	374	5.479358792863787
DeltaCode32	1308.88	525.909	286	5.499787364527583
FAST	576.848	176.602	4298	4.486120236665011
GammaCode16	1295.16	376.979	393	4.555501396209001
GammaCode32	1277.66	461.069	305	4.617146125808358
PGM++128	483.623	96.2237	5	0.6172595618292689
PGM++32	375.941	91.6547	18	0.7323802512139082
PGM++8	407.922	114.468	71	0.9671946777030827
PGM-index128	453.727	96.3062	-	0.6138314576819539
PGM-index32	373.153	95.3587	-	0.7208290092647076
PGM-index8	426.486	102.306	-	0.9702884217724205
PLEX128	324.538	60.9466	14	3.8245395475998523
PLEX32	280.333	48.4523	55	4.365458740666509
PLEX8	263.299	50.5369	176	5.771949747949838
RMI-compact	308.221	34.627	7	0.0009554382413625717
RMI-large	199.821	33.7378	403	0.11063835881650448
Roaring	677.907	44.1218	410	19.318583088554444
SIMD-BTree	213.019	49.9244	1	1.3844092460349202
SIMD-SampledBTree	154.795	31.1628	101	1.1622893745079637
std::vector	754.601	70.9575	1600	-

Table 12: Tabular data: FB uint64 dataset

Compressed Index	Scan 10	Scan 100	Scan 1K	Scan 10K
DeltaCode16	169.992	159.224	157.661	157.319
DeltaCode32	290.61	267.247	264.346	264.193
EliasFano	58.2766	54.9734	54.6018	54.517
GammaCode16	172.952	162.453	160.759	160.208
GammaCode32	296.3	271.827	268.653	268.174
LA-vector10	48.6642	43.1104	42.454	42.316
LA-vector12	44.6708	41.3583	40.7261	40.6656
LA-vector6	48.3177	44.8948	44.1587	43.9621
LA-vector8	45.4154	42.5456	41.6799	41.5476
LA-vectoropt	42.1601	38.6665	37.9378	37.7861
SIMD-BTree	6.66184	5.09499	4.81021	4.67927
std::vector	0.509742	0.195004	0.164005	0.260916

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	182.565	154.161	147.437	146.574
DeltaCode32	288.946	243.979	234.263	233.304
EliasFano	65.4107	54.7323	54.0158	53.7079
GammaCode16	163.564	142.305	138.401	136.443
GammaCode32	265.966	227.669	220.628	219.269
LA-vector10	41.4106	37.558	37.3121	36.8184
LA-vector12	43.0561	39.0688	38.8242	38.2516
LA-vector6	42.0968	36.9566	36.6857	36.2626
LA-vector8	37.9982	33.9595	33.8035	33.3024
LA-vectoropt	69.4947	50.6082	49.0568	48.5937
SIMD-BTree	11.9381	6.87865	5.09214	4.72408
std::vector	2.15582	0.71557	0.508159	0.453078

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	80.2689	68.1329	66.9357	66.298
DeltaCode32	104.653	90.8702	89.3716	88.872
EliasFano	61.545	54.6491	53.9856	53.5898
GammaCode16	80.1803	68.2115	67.147	66.5151
GammaCode32	104.26	90.1701	88.6016	88.061
SIMD-BTree	8.58445	6.35649	5.00639	4.72364
std::vector	1.87378	0.711653	0.494541	0.433225

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	123.492	101.815	99.7216	99.5902
DeltaCode32	179.717	148.469	146.481	146.433
GammaCode16	138.03	116.485	112.413	112.255
GammaCode32	203.892	168.523	165.228	165.047
SIMD-BTree	11.385	10.5871	6.91615	6.60085
std::vector	2.5901	2.01242	1.07582	0.934759

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: 5.59x (books_50M_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-vectoropt: MAX: 101.87x (fb_200M_uint64) - MIN: 10.57x (companynet_uint32)
LA-vector6: MAX: 11.36x (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: 11.04x (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: 2.14x (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 (normal_800M_uint32)
PGM8: MAX: 2.30x (friendster_50M_uint32) - MIN: 1.71x (companynet_uint32)
PGM32: MAX: 2.07x (friendster_50M_uint32) - MIN: 1.71x (companynet_uint32)
PGM128: MAX: 2.02x (fb_200M_uint64) - MIN: 1.57x (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: 3.08x (books_50M_uint64) - MIN: 2.14x (companynet_uint32)
SIMD-SampledBTree: MAX: 3.13x (books_800M_uint64) - MIN: 2.14x (companynet_uint32)
FAST: MAX: 5.68x (books_800M_uint64) - MIN: 3.43x (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 - books_50M_uint64 - existing: First correction too large
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 - books_50M_uint64 - missing: First correction too large
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 - books_50M_uint64 - buildtime: First correction too large
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
```