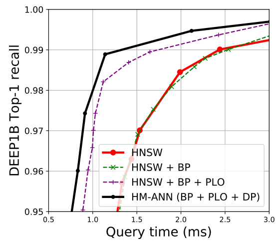

Figure 1: Comparison of techniques

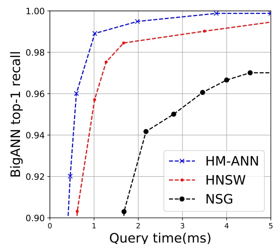

Figure 2: Search perf. on BigANN.

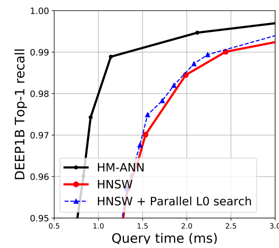

Figure 3: HNSW with parallel L0 search.

The contribution of HM-ANN on billion-scale ANNS. (R2) The core of our work is to show that we can host billion-scale datasets on a single machine with *both* high accuracy and fast speed using HM, outperforming existing solutions. We will revise the claim in the paper to a more accurate one as "a fast and accurate billion-scale nearest neighbor search solution on a single node without compression".

The improvement from data prefetching from slow memory to fast memory. (R2) Figure 1 contains a series of "stepping stones" between HNSW and HM-ANN to show how each optimization of HM-ANN contributes to its improvements, including the data prefetching (DP) asked by the reviewer. "HNSW + Bottom-up promotion (BP)" modifies the HNSW algorithm, mapping the bottom-most layer (i.e., L0) to the slow memory while building a high-quality projection of L0 in fast memory without significantly impacting search efficiency. It provides the benefit of improved search quality in fast memory while providing better entry points to L0 search in slow memory. Together with the parallel L0 search (i.e., "HNSW + Bottom-up promotion (BP) + Parallel L0 search (PL0)") it significantly improves the search efficiency versus running HNSW on HM without explicit data management. For example, to reach a 99% recall target, HM-ANN reduces the query time by 1.75x compared with HNSW. Finally, by prefetching data from slow memory to fast memory, HM-ANN further pushes the search efficiency frontier.

Deployment effort of HM-ANN. (R2) Our performance model has the benefit of significantly pruning the parameter search space that cannot satisfy the response time and accuracy constraints, so it actually expedites the deployment.

The performance gain of HM-ANN on billion-scale datasets. (R2) If we look at the high accuracy range, HM-ANN obtains top-1 recall of >95% and >99% within 0.5 ms and 1.3 ms respectively, which is 2x and 3.3x faster than HNSW. This can be seen from the latency-vs-recall curve with recall larger than 90% for the BIGANN1B dataset in Figure 2.

HM-ANN and SSD storage. (R3) HM-ANN tries to access the data in slow memory, which means HM-ANN does not work with SSD-based storage directly. Combined with SSD, HM-ANN can be used as an in-memory index for hosting even larger datasets. This would make an interesting future study.

Theoretical analysis on the approximation ratio. (R4) We agree it is important to have a theoretical guarantee in terms of the approximation ratio. However, we expect it to be difficult, since all the existing state-of-the-art graph-based NNS algorithms (e.g., HNSW) lack such guarantees. There are multiple paths to advance the state-of-the-art ANN search, and this work focuses on one of them. We can still at least provide a more detailed search time complexity analysis for HM-ANN. HM-ANN constructs each layer as a navigable small world graph, which enables the number of hops scales logarithmically on the greedy search path. Similar to HNSW, HM-ANN constructs the graph with a fixed maximum number of links for each element, which guarantees that the average degree of each element in one layer is constant. The overall number of distance computation is therefore proportional to a product of the number of hops and the average degree of the elements on the greedy path. Therefore, the search complexity in each layer of HM-ANN is logarithmic. Given a layer i with N_i elements, the search complexity of the layer i is $O(log(N_i))$. We then analyze the overall search complexity of HM-ANN. Even with the bottom-up promotion, the maximum number of elements in each layer of HM-ANN remains N. Therefore, the overall search complexity of HM-ANN stays at $O(log(N))$.

HNSW with Parallel L0 search. (R4) We added an experiment to investigate whether it is sufficient to just modify the search procedure without modifying the hierarchical NN graph of HNSW to achieve similar performance gains as HN-ANN. Figure 3 shows the latency-vs-recall performance of default HNSW using parallel L0 search. We use T nearest neighbours found during HNSW L1 search as entry points for the parallel search in L0, where T is the number of parallel threads. We set $T = 4$, same as HM-ANN. HNSW with parallel L0 search only slightly outperforms HNSW. This suggests that parallel L0 search alone is not sufficient for performance improvement. Without it, the elements of L1 in HNSW are selected randomly and sparse, and the entry nodes found through L1 search are sub-optimal. As a result, even though the parallel search in L0 searches more nodes under the same time, the accuracy only slightly improves.

DiskANN evaluation hardware and software settings. (R4) The evaluation platform of HM-ANN and that of DiskANN have the same CPU, and the memory bandwidth and latency of two evaluation platforms are comparable. The software settings are also similar.