RIANN: Real-time Incremental Learning with Approximate Nearest Neighbor on Mobile Devices

Jiawen Liu and Zhen Xie, *University of California, Merced;* Dimitrios Nikolopoulos, *Virginia Tech;* Dong Li, *University of California, Merced*

https://www.usenix.org/conference/opml20/presentation/liu

This paper is included in the Proceedings of the
2020 USENIX Conference on Operational Machine Learning.
July 28–August 7, 2020
978-1-939133-15-1

RIANN: Real-time Incremental Learning with Approximate Nearest Neighbor on Mobile Devices

Jiawen Liu[†], *Zhen Xie*[†], *Dimitrios Nikolopoulos*[‡] , *Dong Li*[†]
[†]University of California, Merced [‡]Virginia Tech

Abstract

Approximate nearest neighbor (ANN) algorithms are the foundation for many applications on mobile devices. Real-time incremental learning with ANN on mobile devices is emerging. However, incremental learning with current ANN algorithms on mobile devices is hard, because data is dynamically and incrementally generated and as a result, it is difficult to reach high timing and recall requirements on indexing and search. Meeting the high timing requirements is critical on mobile devices because of the requirement of short user response time and because battery lifetime is limited.

We introduce an indexing and search system for graph-based ANN on mobile devices called RIANN. By constructing ANN with dynamic ANN construction properties, RIANN enables high flexibility for ANN construction to meet the strict timing and recall requirements in incremental learning. To select an optimal ANN construction property, RIANN incorporates a statistical prediction model. RIANN further offers a novel analytical performance model to avoid runtime overhead and interaction with the device. In our experiments, RIANN significantly outperforms the state-of-the-art ANN (2.42$\times$ speedup) on Samsung S9 mobile phone without compromising search time or recall. Also, for incrementally indexing 100 batches of data, the state-of-the-art ANN satisfies 55.33% batches on average while RIANN can satisfy 96.67% with minimum impact on recall.

1 Introduction

Approximate nearest neighbor (ANN) is an essential algorithm for many applications, e.g., recommendation systems, data mining and information retrieval [2, 4, 7, 10, 15–17] on mobile devices. For example, applications on mobile devices often provide recommendation functionalities to help users quickly identify interesting content (e.g., videos from YouTube [6], images from Flickr [14], or content from Taobao [9]). To meet user requirements, it is important to incrementally construct ANN on mobile devices in real-time. For example, it is essential to recommend to users new content that is of interest while the content is still fresh, as there is a clear tendency for users to prefer newer content. This necessitates real-time incremental learning for ANN on mobile devices.

However, real-time incremental learning for ANN on mobile devices imposes two challenges. First, current ANN models cannot meet the real-time requirement of high recall for incremental learning due to static graph construction. Specifically, with different size of batches in incremental learning, current ANN algorithms either index batches of data with high recall without meeting the real-time requirement or index data in real-time with low recall.

Second, current ANN algorithms perform end-to-end indexing hence indexing time, query time and recall are unknown to users prior to or during ANN indexing, while these results are required to reach high recall in real-time incremental learning.

To address the above challenges, we propose RIANN, a system to enabling real-time ANN incremental learning on mobile devices. To achieve our goals, we propose a dynamic ANN indexing structure based on HNSW [13]. With the dynamic ANN graph construction, we can target different indexing times, query times and recall on the fly. Next, we propose a statistical performance model to guide dynamic ANN construction over millions of possible properties. We further propose an analytical performance model to avoid interaction with mobile devices and runtime overhead.

2 Framework Design

Dynamic ANN graph construction. Currently, most graph-based ANN algorithms work in the following manner: during graph construction, the algorithms build a graph $G = (P, E)$ based on the geometric properties of points in dataset P with connecting edges E between the points. At search time, for a query point $p \in P$, ANN search employs a natural greedy traversal on G. Starting at some designated point $d \in P$, the algorithms traverse the graph to get progressively closer to p. The state-of-the-art ANN algorithm HNSW exploits the above procedure with building a multi-layer structure consisting of hierarchical set of proximity graphs (layers) for nested subsets of the stored elements.

However, since current graph-based ANN algorithms including HNSW are designed using static graph construction properties, there is little flexibility in controlling graph construction properties for real-time ANN incremental learning.

To address this problem, we propose RIANN to construct graphs in a dynamic manner. The dynamic graph construction depends on user requirements and a batch size of data

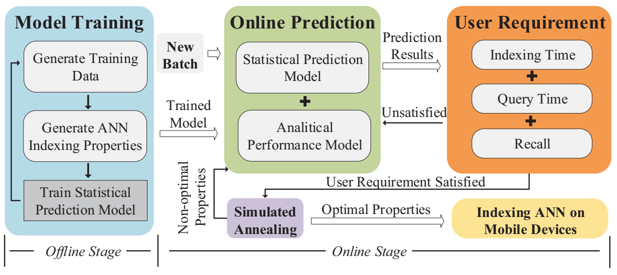

Figure 1: Overview of RIANN.

points. To achieve this goal, we build dynamic construction graph properties (e.g., out-degree edges of each point and candidates to build those edges). The advantage of dynamic group construction properties is that we can meet the real-time requirement while maintaining high recall.

Domain-specific statistical prediction model. The traditional approach to obtain the optimal indexing properties is to examine different indexing properties [3, 9, 11, 13]. However, to obtain the optimal indexing properties, this approach 1) requires an excessive amount of time (days and even weeks) [3] to obtain the optimum and 2) requests the exact indexing data size which is impractical in ANN incremental learning.

We propose a statistical prediction model to solve the problem. Figure 1 presents the overview of our design. The statistical prediction model is to estimate the recall and indexing or query time of each construction property for indexing a batch of data. The model is based on gradient boosted trees [8](GBTs) with simulated annealing [12]. We use XGBoost [5] as the GBTs model for training and implement a light-weighted XGBoost inference engine for mobile devices.

Analytical performance model. Though the prediction model is promising to predict ANN recall, the model has two issues to predict indexing/query time: 1) it interacts with mobile devices frequently to collect training data and 2) it incurs nonnegligible runtime overhead.

To address those problems, we propose an analytical performance model to predict indexing/query time at runtime. Equation 1 is used to estimate the time of querying data point $p \in P$ in one layer, where P refers to the set of points in one batch. The metric is defined as:

$$T_{lyr}(cand, deg, N) = T_d * h(cand, D_{avg}(deg, N)) * D_{avg}(deg, N) \quad (1)$$

Where *cand* represents candidate points, *deg* is the out-degree, N is the set of all points in one layer and T_d represents the time to calculate the distance. $h(cand, D_{avg}(deg, N))$ is a function to obtain the average number of hops from the entry point to the target point. The function can be formulated with small sample profiling offline. $D_{avg}(deg, N) = \frac{1}{|N|+1}(\sum_{i=1}^{|N|} N_i + deg)$ calculates the average out-degree after inserting p.

With the number of candidates and out-degree of p, the query time T_{qry} and indexing time T_{idx} are defined as follows:

$$T_{qry}(cand_q, deg) = \sum_{\ell=2}^{\ell_{max}} T_{lyr}(1, deg, N_\ell) + T_{lyr}(cand_q, deg*2, N_1) \quad (2)$$

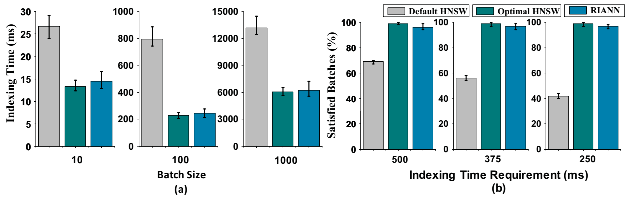

Figure 2: (a) Indexing time of RIANN and HNSW with different batch size. (b) Percentage of satisfied batches of RIANN and HNSW with different indexing time requirement.

$$T_{idx}(cand_i, deg) = \sum_{\ell=\ell_p}^{\ell_{max}} T_{lyr}(1, deg, N_\ell) + \sum_{\ell=2}^{\ell_p} (T_{lyr}(cand_i, deg, N_\ell) + f(deg)) + T_{lyr}(cand_i, dre*2, N_1) + f(dre) \quad (3)$$

Where ℓ_{max} is the maximum number of layers, ℓ_p represents the indexing layer for p and $cand_q$ and $cand_i$ represent the number of candidates for query and indexing. The time of querying p from ℓ_{max} to ℓ can be calculated by $\sum_{\ell}^{\ell_{max}} T_{lyr}(cand, deg, N_\ell)$. The time of updating out-degree of p is calculated by $f(deg)$ that depends on the implementation and linear to the number of out-degree.

3 Evaluation

Comparison of RIANN and HNSW. We compare RIANN with HNSW which is the state-of-the-art ANN algorithm [3]. We employ SIFT [1] dataset on a Samsung S9 with Android 9. We use the graph construction properties listed in the paper [13] denoted as default HNSW. The optimal HNSW represents hypothetical results in which 1) users know the data size of ANN indexing which is impractical; 2) users spend a large amount of time (days and even weeks) to obtain the optimal HNSW. We experiment different batch size (10, 100 and 1000) in batch increments of 10 for incremental learning.

In Figure 2, we observe that RIANN shows significant performance improvement (2.42 times speedup on average) than the default HNSW and 8.67% performance less than the optimal HNSW while RIANN maintains the same recall and query time compared to the optimal and default HNSW. The runtime overhead of RIANN is included in Figure 2.

Evaluation with User Requirement. We incrementally index 100 batches and the batch size starts from 10 to 1000. In Figure 2, we observe that 55.33% batches are satisfied using default HNSW while RIANN can satisfy 96.67% with only 2.43% loss in recall.

4 Conclusion

We present RIANN, a real-time graph-based ANN indexing and search system. It constructs graphs in a dynamic manner with a statistical prediction model and an analytical performance model to incrementally index and search data in real-time on mobile devices. RIANN significantly outperforms the state-of-the-art ANN (2.42× speedup) without compromising query time or recall in incremental learning. Also, for incrementally indexing 100 batches of data, the state-of-the-art ANN satisfies 55.33% batches on average while RIANN can satisfy 96.67% with compromising 2.43% recall.

References

[1] Laurent Amsaleg and Hervé Jegou. Datasets for approximate nearest neighbor search. http://corpus-texmex.irisa.fr.

[2] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces. *Proceedings of the VLDB Endowment*, 11(8):906–919, 2018.

[3] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87:101374, 2020.

[4] Lei Chen, M Tamer Özsu, and Vincent Oria. Robust and fast similarity search for moving object trajectories. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 491–502, 2005.

[5] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.

[6] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, et al. The youtube video recommendation system. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 293–296, 2010.

[7] Arjen P de Vries, Nikos Mamoulis, Niels Nes, and Martin Kersten. Efficient k-nn search on vertically decomposed data. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 322–333, 2002.

[8] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[9] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proceedings of the VLDB Endowment*, 12(5):461–474, 2019.

[10] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 9(1):1–12, 2015.

[11] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[12] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598):671–680, 1983.

[13] Yury A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

[14] Börkur Sigurbjörnsson and Roelof Van Zwol. Flickr tag recommendation based on collective knowledge. In *Proceedings of the 17th international conference on World Wide Web*, pages 327–336, 2008.

[15] George Teodoro, Eduardo Valle, Nathan Mariano, Ricardo Torres, Wagner Meira, and Joel H Saltz. Approximate similarity search for online multimedia services on distributed cpu–gpu platforms. *The VLDB Journal*, 23(3):427–448, 2014.

[16] Chong Yang, Xiaohui Yu, and Yang Liu. Continuous knn join processing for real-time recommendation. In *2014 IEEE International Conference on Data Mining*, pages 640–649. IEEE, 2014.

[17] Yuxin Zheng, Qi Guo, Anthony KH Tung, and Sai Wu. Lazylsh: Approximate nearest neighbor search for multiple distance functions with a single index. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2023–2037, 2016.