Learned Adaptive Accuracy-Cost Optimization for Machine Learning Systems

Conglong Li

CMU-CS-20-105
May 2020

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
David G. Andersen, Chair
Yuxiong He, Microsoft AI and Research
Michael Kaminsky
Rashmi K. Vinayak

Submitted in partial fulfillment of the requirements
for the Degree of Doctor of Philosophy.

This research was sponsored by the National Science Foundation under grant numbers CCF-1535821, CNS-1314721, and CNS-1700521, Google, Vanguard Charitable, and the VMware University Research Fund. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Machine Learning, Information Retrieval, Recommendation Systems, Cache, Similarity Search, Approximate Nearest Neighbor Search, Sponsored Search, Search Advertising, Workload Analysis

Abstract

This dissertation seeks to address the challenge of making adaptive accuracy-cost balancing inside systems for large-scale machine learning-based recommendation services. We show that it is important to make performance tradeoff decisions at a per-query basis instead of a predefined policy for all queries. We show that we can achieve a better tradeoff between accuracy and cost by leveraging lightweight machine learning models to make more adaptive decision-making inside systems infrastructure.

Large-scale recommendation services have two computation-heavy components with strict accuracy and latency targets: scoring (typically achieved by complex machine learning models) and candidate retrieval (typically achieved by approximate nearest neighbor search). We first introduce a caching system for scoring component in recommendation systems (in particular search advertising systems). Inside the cache, we leverage lightweight machine learning models to make adaptive cache refresh decisions, which provides a better balance between recommendation accuracy and computation cost. This leads to a better net profit in the search advertising context. We then present the learned adaptive termination for approximate nearest neighbor search inside the candidate retrieval component. We leverage lightweight machine learning models to decide how much to search for each query, which provides a better balance between the search accuracy and latency (computation cost).

Acknowledgments

I am very fortunate to have David G. Andersen as my advisor. Dave's passion on all kinds of computer science problems always motivates me to think deeper and work harder. His great sense of humor and blogs about life lessons always make people smile and enlightened. And I truly appreciate his support for the collaboration with Microsoft.

I am very fortunate to have Yuxiong He as my thesis committee member and as my 4-year-long unofficial co-advisor. Yuxiong sponsored my first internship and became my mentor in summer 2016. Then we continued our collaboration for 4 years, including 3 internships. As an industrial researcher, Yuxiong provided invaluable motivation and guidance for the works in this dissertation.

Michael Kaminsky and Rashmi K. Vinayak have provided valuable feedback to make this dissertation possible to exist. I also greatly appreciate Michael for his mentoring on the network switch scheduling project during the early years of my PhD program.

I appreciate all the colleagues I worked with at Microsoft: Minjia Zhang, Sameh Elnikety, Qiang Fu, Ramu Movva, Chao Li, Murat Ali Bayir, and many others. They provided many feedback and guidance to my research.

I want to thank my colleagues and friends at CMU for making my life enriching and fun: Huanchen Zhang, Anuj Kalia, Dong Zhou, Hyeontaek Lim, Thomas Kim, Sol Boucher, Christopher Canel, Giulio Zhou, Angela Jiang, Daniel Wong, Charlie Garrod, Yingjie Zhang, and many others. I also want to thank the staffs at CMU for their support: Deborah Cavlovich, Angy Malloy, Karen Lindenfelser, Joan Digney, Kathy McNiff, Patricia S. Loring, and many others.

Finally, I would like to thank my parents and all my family members for their unwavering support throughout the years of my study.

This dissertation includes and extends prior published work:

1. Conglong Li, David G. Andersen, Qiang Fu, Sameh Elnikety, and Yuxiong He. Workload Analysis and Caching Strategies for Search Advertising Systems. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC)*, 2017 [70].
2. Conglong Li, David G. Andersen, Qiang Fu, Sameh Elnikety, and Yuxiong He. Better Caching in Search Advertising Systems with Rapid Refresh Predictions. In *Proceedings of the 2018 World Wide Web Conference (WWW)*, 2018 [72].

3. Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2020 [73].

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Today, we see the deployment of an increasingly large number of machine learning systems that apply powerful, expensive computation to produce important yet approximate answers. In these settings, it is common that developers desire to simultaneously optimize for generally-conflicting objectives: accurate answers and low latency/computational cost to produce the answers.

In this dissertation, we take a systems approach to balancing these accuracy/cost tradeoffs for machine learning systems. While important exceptions exist, many common machine learning approaches to reduce cost do so by creating entirely new approaches or structures, and thus, cannot be modularly applied. Again with important exceptions, modular systems approaches such as caching often are not designed to work with systems that are approximate and noisy.

For machine learning systems and query processing systems in general, queries tend to have diverse accuracy-cost properties and requirements: It could take different amount of work (cost) to reach the same accuracy for each query. On the other hand, different queries may have different accuracy requirements (e.g., depending on the penalty of low accuracy), which means same amount of work could provide different benefits. In this dissertation, we take a new approach to balancing the cost/benefit tradeoff inside systems by asking: "how much (more) work should be done for this query?", and training lightweight machine learning models to help answer this question.

In this dissertation, we focus generally on the important case of recommendation systems. Recommendation systems are a kind of information filtering system that seeks to predict the "rating" or "preference" a user would give to an item [93]. They play a critical role in a wide range of web applications including online shopping (e.g., Amazon, Ebay), web search (e.g., Google Search, Bing Search), multimedia services (e.g., YouTube, Netflix, Spotify), social networks (e.g., Facebook, Twitter), and advertising (e.g., Google AdWords, Bing Ads, Facebook Ads).

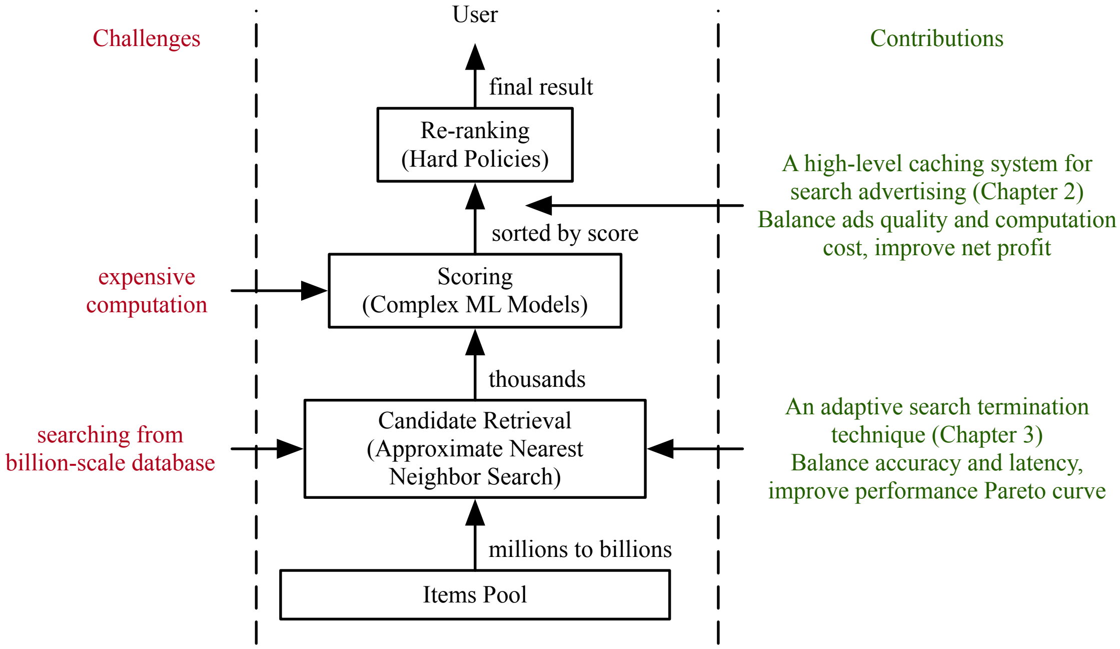


Figure 1.1: Simplified overview of recommendation systems, together with the challenges and our contributions.

Large-scale recommendation systems commonly consist of 3 components as illustrated in Figure 1.1. This is similar to cascading classifiers [38], where each component further refines the query result. First, the candidate retrieval component selects a subset of relevant items from the whole items pool. This could mean filtering from billions of items to thousands of items. Given the enormous number of items in the pool, it is important to balance the accuracy and latency for candidate retrieval. Besides classic approaches such as exact keyword matching, one popular candidate retrieval technique is approximate nearest neighbor search, which leverages the embedding vectors of queries and items and ranks the items by distances to the query [27].

Second, the scoring component scores and ranks the candidates based on a predicted relevance score. Since this component evaluates a relatively small subset of items, the system can use a more precise model relying on additional feature. Nowadays we incorporate different complex machine learning models to do the scoring. Finally there is a re-ranking component to apply some hard rules and to ensure diversity, freshness and fairness.

Advances in machine learning enable increasingly accurate recommendation services. However, web applications are latency-sensitive where the latency budget is usually within one or a few seconds. In addition, in the context of most recommendation services, it is also important to save computation cost. Thus we need a balance between accuracy and operating cost, which becomes more and more challenging: For the candidate retrieval stage, it is hard to balance the accuracy and latency as the database keeps growing to billion-scale. For the scoring stage, the advanced complex machine learning models require expensive computations.

This dissertation identifies a modular approach to balancing accuracy/cost tradeoffs for important classes of machine learning problems that can be applied easily to different backend algorithms. We apply these techniques to improve the efficiency of the candidate retrieval and scoring components inside recommendation systems, since both of them contribute to a great portion of service latency/operating cost.

This dissertation provides evidence to support the following statement:

Thesis Statement: *In real-world recommendation services, answers are approximate and expensive. Lightweight machine learning models can be used for adaptive decision-making inside systems to better trade between accuracy and cost.*

We have completed the following contributions for this dissertation:

C1. A workload analysis of one kind of recommendation systems: search advertising systems. We show that modern recommendation systems introduce expensive computation, but there is an opportunity to use caching to save it. On the other hand, caching may introduce penalties in terms of accuracy/quality drop of recommendation results and revenue loss in advertising context. It is important to adaptively decide when to use the cached result (e.g., only for the queries with lower potential revenue loss). (Chapter 2)

C2. A high-level caching system for recommendation systems (in particular search advertising systems) to reuse the expensive scoring results. Inside the cache, we leverage lightweight machine learning models to make adaptive cache refresh decisions. This leads to a better balance between recommendation accuracy and computation cost, which eventually improves the net profit in the search advertising context. (Chapter 3)

C3. A performance analysis of state-of-the-art approximate nearest neighbor search techniques (for the candidate retrieval component). We show that different search queries require different amount of search to find the true nearest neighbor, but existing ANN search approaches are not able to exploit this variance. (Chapter 4)

C4. An adaptive search termination technique for approximate nearest neighbor search. We leverage lightweight machine learning models to decide how much to search for each query, which provides a better balance between accuracy and latency. (Chapter 5)

1.1 Caching strategies for recommendation systems

Caching is commonly used to reduce computation cost by reusing previous results. It is often easy to implement, fast, and modular: caches can be reused in many situations. Unfortunately, since recommendation query answers are approximate and noisy, caching will also reduce the accuracy of the answers which may introduce various penalties for different queries. For each query, caching is only desirable if the cost saving benefit is larger than the penalty. In the context of this, we present a technique that leverages lightweight machine learning models inside the cache to make adaptive decisions to balance the cost saving reward and accuracy drop penalty. This is the first of the two situations to which we apply our model of asking whether to stop working (use the cached result) or do more work to produce a better answer. In this dissertation we focus on one kind of recommendation systems, the searching advertising systems. But we believe that our findings are also applicable to other recommendation systems that need to balance the reward and penalty of caching.

To maximize profit and connect users to relevant products and services, search advertising systems use sophisticated machine learning algorithms to estimate the revenue expectations of thousands of matching ad listings per query [46, 50, 82]. These machine learning computations constitute a substantial part of the operating cost, e.g., 10% to 30% of the total gross revenues. It is desirable to cache and reuse previous computation results to reduce this cost, but caching introduces approximation which comes with various revenue loss for each query. To maximize cost savings while minimizing the overall revenue impact, an adaptive refresh policy is required to decide when to refresh the cached computation results.

We first design, implement, and evaluate a heuristic-based cache design which uses revenue history to assign different refresh policies for each query. We then improve upon this baseline heuristic approach by using a lightweight machine learning model to make the same caching decisions. This approach uses gradient boosted regression trees [35] that integrate a richer set of features, and we show that it attains better accuracy with low additional cost.

We evaluate the heuristic-based and prediction-based cache designs by simulations on production traces from Bing Ads, a major commercial search advertising system. A traditional cache with fixed refresh rate can reduce cost by up to 30.7% while having negative revenue impact as bad as -5.16%. In comparison, the proposed heuristic-based cache design can reduce cost by up to 17% with a negative revenue impact at -0.29%. The proposed prediction-based cache can reduce cost by up to 24% while capping negative revenue impact at -0.02%. Based on Microsoft's earnings release for FY16 Q4, the traditional cache would introduce a net profit impact between $\$-30.6$ and $\$59.4$ million in the quarter. The proposed heuristic-based cache would increase the net profit of Bing Ads by $\$20.7$ to $\$70.5$ million in the quarter, and the prediction-based cache could increase the net profit by $\$35.2$ to $\$106.1$ million.

1.2 Learned adaptive early termination for approximate nearest neighbor search

Since it is not always possible to use cache to avoid all computations (e.g., queries with high revenue expectation in search advertising), we next focus on applying our core approach to one of the key underlying components. For the candidate retrieval component inside recommendation systems, the problem of identifying a set of "similar" real-valued vectors to a query vector plays a critical role. However, retrieving these vectors and computing the corresponding similarity scores from a large database is computationally challenging. Approximate nearest neighbor (ANN) search relaxes the guarantee of exactness for efficiency by vector compression and/or by only searching a subset of database vectors for each query. Searching a larger subset increases both accuracy and latency. State-of-the-art ANN approaches use fixed configurations that apply the same termination condition (the size of subset to search) for all queries, which leads to undesirably high latency when trying to achieve the last few percents of accuracy. We find that due to the index structures and the vector distributions, the number of database vectors that must be searched to find the ground-truth nearest neighbor varies widely among queries. Critically, we further identify that the intermediate search result after a certain amount of search is an important runtime feature that indicates how much more search should be performed.

To achieve a better tradeoff between latency and accuracy, we propose a novel approach that adaptively determines search termination conditions for individual queries. To do so, we build and train gradient boosting decision tree models and neural network models to learn and predict when to stop searching for a certain query. These models enable us to achieve the same accuracy with less total amount of search compared to the fixed configurations. This is the second of the two situations to which we apply our model of asking how much work should be done for each query. We apply the learned adaptive early termination to state-of-the-art ANN approaches, and evaluate the end-to-end performance on three million to billion-scale datasets. Compared with fixed configurations, our approach consistently improves the average end-to-end latency by up to 9.4 times faster under the same high accuracy targets. Our approach is open source at github.com/efficient/faiss-learned-termination.

Chapter 2

Search Advertising Systems Workload Analysis

In this chapter, we present workload analysis of one kind of recommendation systems, the searching advertising systems, as a motivation for the proposed caching system. Sponsored search advertising is an indispensable part of the business model of modern web search engines. For a given search query from a user, the search advertising system presents several related sponsored search results (advertisements) together with the general search results from the web search engine. These advertising systems usually adopt a pay-per-click model that advertisers are charged only if their advertisements are clicked by a user.

Search advertising has become a market of tens of billions of dollars per year. Search advertising publishers, such as Google AdWords and Bing Ads, aim to accurately connect users with products and services matching their interests, and earn revenue from advertisers. To better predict user behavior and maximize ad click revenue, search advertising systems estimate the expected revenue of thousands of matching ad listings per query using various machine learning algorithms [46, 50, 82].

Machine learning-based ads scoring provides accurate matchings between users and advertisers. However, as the number of users, ad candidates, and features increase, the dollar cost of machine learning computations becomes an increasing portion of the cost of search advertising systems [46]. We study one week of production traces, with billions of queries, from Bing's advertising system. Workload analysis shows that the machine learning algorithms occupy hundreds of machines for tens of milliseconds to select the ads for each query. Caching the computation results (list of selected ads) of machine learning algorithms could reduce the amount of computation for processing ads, thus reducing infrastructure cost and potentially improving the net profit.

For the rest of this chapter, Section 2.1 describes the basic concept of search advertising systems. Section 2.2 presents related works. Section 2.3 presents the workload analysis.

2.1 Search advertising systems

Search advertising, or sponsored search, is an ecosystem including three participants: users, advertisers, and publishers. Users search for keywords trying to get relevant and qualitative search results; advertisers set bidding budget on their interested keywords to get the chance for showing their own ads to find customers and boost sales. Publishers, such as Bing Ads and Google AdWords, bridge the two by renting out space on search result page to show ads. Nowadays publishers usually adopt a pay-per-click model that advertisers only pay when their ads get clicked by users. Since the space to show ads is limited, publishers need to select a few ads from the ads pool that contains all advertisers' ads. To create values for all participants of the ecosystem, search advertising systems need to select ads that are semantically relevant to the search query, qualitative, and most profitable.

Figure 2.1 plots the simplified workflow of Bing Ads as an example to show how a search advertising system serves ads to search queries, and how the proposed cache works. There are mainly three components in Bing Ads system: the initial candidate retrieval, the scoring-based selection, and the final auction. Advertisers provide Bing Ads their ad listings together with the bidding budgets, targeting keywords/user groups, and various constraints (spatial, temporal, contextual). These data are stored in an ads pool partitioned across hundreds of servers.

When an user search query arrives, the initial candidate retrieval selects ads with matching targeting keywords and user groups (e.g., location, gender, etc.) from the ads pool. This matching process is parallelized to satisfy the throughput and latency requirements. For the keyword matching process, Bing Ads provides both exact matching and approximate matching that leverages different machine learning algorithms (including approximate nearest neighbor search) to match similar keywords so that advertisers are able to show their ads to more users. There are normally thousands of ads selected from the candidate retrieval.

Second, the scoring component scores candidate ads and selects tens of them with the highest scores. The score depends on both the advertiser's bidding budget and the ad's quality. Bing Ads measures ad quality by three factors: the predicted click-through rate, the ad relevance, and the landing page (the webpage pointed by the ad) experience. Click-through rate represents the click probability. Ad relevance represents the relevance between the ad, the search query, and the user. As different users may have different behaviors, personalization information such as the gender, age, and location of the user are important factors. The landing page experience represents the likelihood of the user to

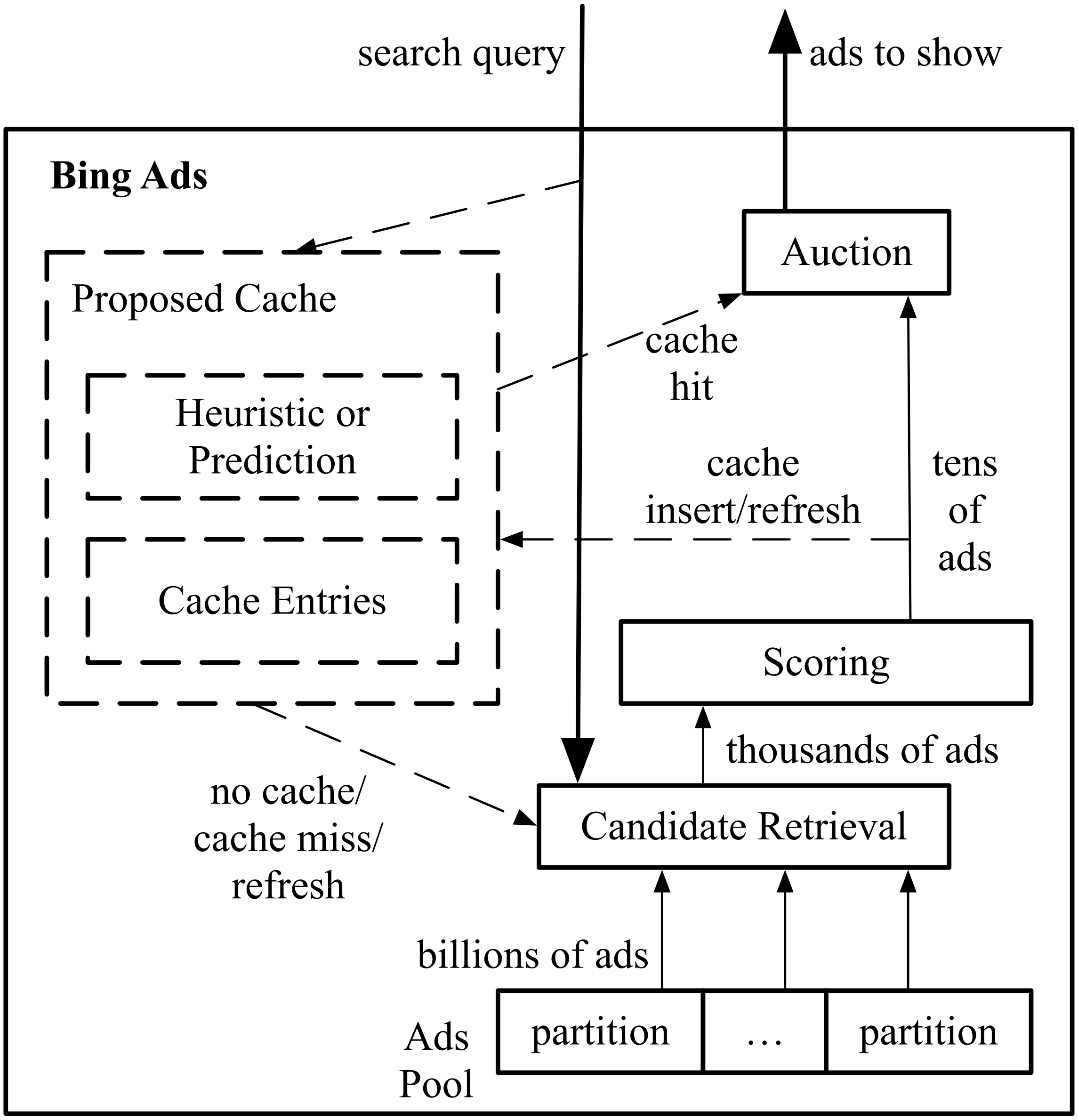


Figure 2.1: Simplified workflow of how Bing advertising system serves ads to users.

get good experience on the landing page. Due to the complexity of the scoring function and the varied user behaviors, search advertising systems leverage different complex learning algorithms to optimize this scoring component [46, 50, 82].

Last, tens of ads with the highest scores are sent to the final auction process for re-ranking. The auction determines the position of each ad, and how much will be charged when an ad is clicked by the user. This process is always required, as advertisers' ad listings and bidding budgets change dynamically.

Due to the huge input size and the number of involved features, both the candidate retrieval and scoring components consume enormous computation cost. To reduce this cost, we propose to build an ad-serving cache between the scoring and the final auction. The cache is essentially a key-value store where the key is the query phrase (optionally combined with other features) and the value is the selected ads from previous scoring computation results. On cache miss, ads are selected from the candidate retrieval and scoring-based selections and then inserted into cache. On cache hit, the cache first uses the heuristic or prediction framework to decide whether or not refresh the cache entry. If the heuristic decides not to refresh or the prediction value is lower than a threshold, the cached ads are sent to the final auction. Otherwise the cache refreshes the cached ads just as the cache miss case.

2.2 Related work

2.2.1 Caching for web search

The closest related work to ours are prior studies on caching systems for web search engines. Since caching for advertising systems can directly affect revenue, simply applying web search engine caching designs would result in huge revenue loss. However, these caching techniques for web search engines inspire the design of our caching designs for advertising systems. To the best of our knowledge, this work is the first to propose cache design for search advertising systems.

Cost-aware and feature-aware caching

Gan *et al.* focus on weighted caching and feature-based caching for web search engines [39]. Their study shows that the processing costs of queries can vary significantly and query traces have application-specific features that is not exploited by previous cache eviction policies. Ozcan *et al.* incorporate the query processing cost into the caching policies and results show that cost-aware policies improve the average query execution time [86]. Cambazoglu *et al.* show that regionalization improve the relevance of the web search results but decreases the hit rates of web search result cache [18]. Sazoglu *et al.* propose to take the hourly electricity prices into account when computing the processing cost of queries [96].

Search advertising systems require consideration of cost and features as well, but the definitions are different. Search advertising caching system has to consider both the processing cost saving and the potential revenue loss when serving the sub-optimal results from cache. In addition to regional features, search advertising systems incorporate diverse features such as revenue history and user's gender and age. These differences require different cost-aware and feature-aware caching strategies.

Refresh policy

Cambazoglu *et al.* argue that caching for large-scale search engines should be able to cope with freshness of the index [19]. They propose to use a time-to-live (TTL) value to invalidate cache entries, and leverage idle back-end cycles to refresh cache entries. Bortnikov *et al.* propose to use an adaptive TTL per cache entry based on the access frequency and a ranking score [16]. Alici *et al.* propose to use generation and update timestamps to estimate the staleness of search query results [5]. Bai *et al.* rely on a subindex of recent changes to the search index to invalidate the stale cache entries [11]. Instead of a fixed TTL value, Alici *et al.* propose to use an adaptive TTL value on a per-query basis to improve the result freshness and reduce the refresh cost [6]. Since minimizing revenue loss is one of the primary goals of search advertising caching, our proposed ads cache needs an unique adaptive refresh policy based on potential revenue loss of each cache entry.

Hybrid caching strategy

Fagni *et al.* propose a hybrid result caching design where the statistically high-frequency queries are stored in a static cache and other queries are stored in a dynamic cache [32]. Baeza-Yates *et al.* study the tradeoff of different caching designs for web search engines, such as static vs. dynamic caching, and caching query results vs. caching posting lists [10]. Ozcan *et al.* propose a hybrid result caching strategy to exploit the tradeoff between the hit rate and the average query response latency [87]. Our workload analysis shows that search advertising systems is also a diverse environment with skewed frequencies and revenue expectations. This shows that a fixed caching policy is not sufficient.

The related works above in web search consider similar features (e.g., cost and user information) and the same intuition about reducing cache staleness compared to caching systems for search advertising. However, since the business models of web search and sponsored search are quite different, these cache designs do not consider revenue-related performance thus cannot be directly applied to search advertising systems.

2.2.2 Prediction framework for sponsored search and web search

Our work differs from frameworks in which machine learning algorithms are used to predict the click-through rate and other performance metrics of the candidate ads in search advertising systems [46, 50, 82]. These prediction frameworks aim to maximize the ad click revenue without considering the computation cost of each prediction. Compared to these prediction frameworks, our work aims to find the best tradeoff decisions between cost savings and revenue impact in order to maximize the net profit of the whole system. To do so, our prediction framework has different performance requirements in terms of accuracy and latency overhead.

In web search, recent works study fast prediction frameworks to predict the execution time of search queries and assign different parallelization decisions based on the predictions [59, 63, 77]. These prediction frameworks enable the web search engine to accurately predict the long-running queries and reduce the tail latency by parallelization. These works motivate us to use rapid machine learning algorithms to solve the caching problem for search advertising systems. Since the performance objectives (latency reduction v.s. profit improvement) are different, our prediction framework has different design in terms of prediction objective and feature selections compared to those recent works in web search.

2.3 Workload analysis

This section describes the workload analysis of the Bing advertising system. We analyze the Bing advertising system logs corresponding to a slice of the whole traffic from Mon Jun 5th 2017 to Sun Jun 11th 2017, which contains billions of queries (hundreds of millions each day). We consider three key personalization features in our workload analysis: location, gender, and age of the user. Although some sensitive numbers are normalized, the workload analysis indicates many opportunities and challenges of caching for search advertising systems.

2.3.1 Performance metrics

To quantify the workload of the Bing advertising system, we use the following performance metrics:

Ad click revenue As described in Section 2.1, a search advertising system takes each user search query, selects the list of ads to show on the web page, and charges the corresponding advertiser when the user clicks one of the shown ads. We call these charges on clicks as ad click revenue of the advertising system.

Ad-serving cost indicator To illustrate the cost of the candidate retrieval and scoring-based selection in Figure 2.1, we use the input size of scoring-based selection (i.e., number

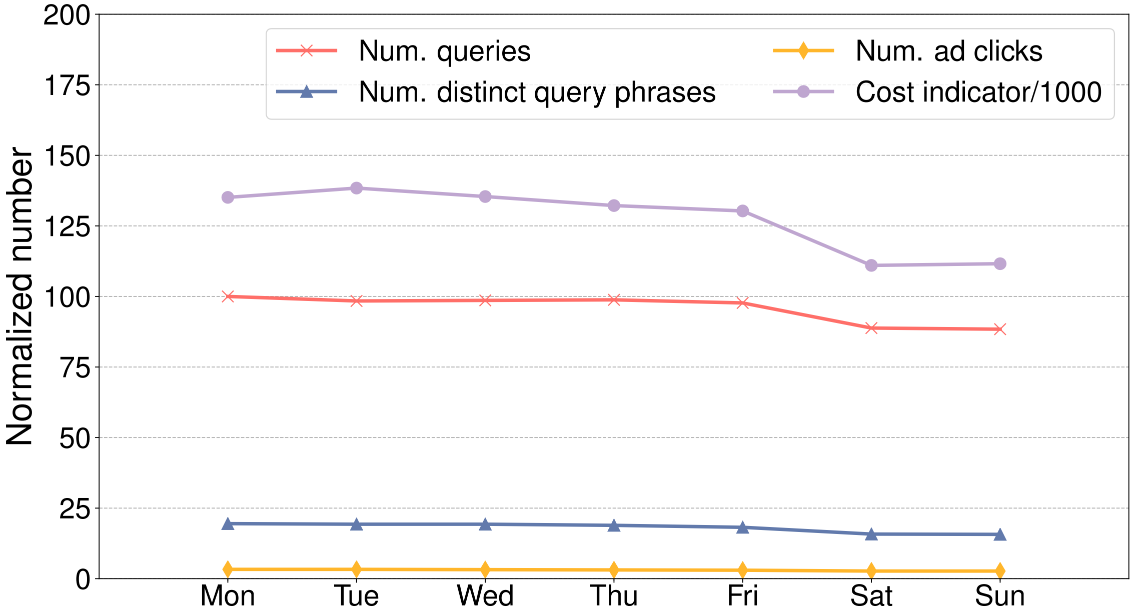


Figure 2.2: Daily normalized statistics of Bing advertising system in a week in US area.

of candidate ads that we need to run complex machine learning predictions) as the cost indicator. Higher cost indicator means more advertisements to be considered by the learning algorithms, thus the computation cost will increase as well.

2.3.2 The workload in a week

To illustrate the general daily statistics in a week, Figure 2.2 plots four daily statistics normalized proportionally: total number of search queries, total number of distinct query phrases, total number of ad clicks, and total sum of cost indicator (divided by 1000). All the numbers in the figure are normalized by multiplying the same constant coefficient.

Among the logs we analyze, the Bing advertising system receives hundreds of millions of queries each day. This total number of queries stays stable among the weekdays and slightly decreases in the weekends. All the other three statistics have similar trends in the week. The total number of distinct query phrases shows that everyday each distinct query phrase has about 5 queries on average. As we will show in the following analysis, the frequency distribution of the query phrases is highly skewed, and the frequencies of top phrases are much higher than 5.

The total number of ad clicks is much smaller compared to the total number of queries. Only about 3% of queries end up with ad click, which means that 97% of learning computations result in no revenue. A recent study shows that the average actual click-through rate per ad is 1.91% among 2367 Google AdWords advertisers, which is similar to our workload [1]. As we will show in the following analysis, most of the ad clicks and the corresponding revenue are contributed by a few percent of distinct query phrases. This motivates us to consider an adaptive caching strategy depending on the revenue history.

In Figure 2.2 the cost indicator is divided by 1000 in order to plot all the lines at the same magnitude. This cost indicator is more than 1000 times of the total number of queries, which means that the scoring-based selection needs to score and select from more than 1000 matching candidate ads on average for each query. This illustrates the learning computation cost per query.

In the following workload analysis, we provide a deeper study of the logs in a single day on Wed Jun 7th, which is the same day we use for simulation evaluation in Section 3.3. We do perform the same analysis on all the days in the week, and the results are similar among different days just like the trend in Figure 2.2.

2.3.3 Frequency distribution

Figure 2.3 plots the percentage of total queries contributed by the top x% query phrases, both with and without personalization information (location, gender, age). When considering the personalization, queries with the same query phrase but different personalization are separated into different categories. As the figure shows, the frequency distribution is highly skewed in both cases. When personalization is not considered, top 1% query phrases contribute to 64% of the total queries. When personalization is considered, the distribution is less skewed since query phrases are separated by different personalization information. However, top 1% query phrases still contribute to as high as 44% of the total queries. This highly skewed frequency distribution indicates that even a small cache could achieve a rather high hit rate, demonstrating an opportunity for caching.

2.3.4 Revenue-related features

Average revenue history per query phrase

For both heuristic-based and prediction-based cache designs, it's important to include revenue-related features into consideration. The historical average ad click revenue for each distinct query phrase is a good candidate, since it indicates the potential revenue for queries with the same phrase. Figure 2.4 illustrates the CDF of average revenue for each query phrase. Due to business confidentiality, the average revenue numbers in the figure are normalized by multiplying the same constant coefficient. Only 1.6% of the distinct query phrases have ads clicked. Moreover, the largest average revenue (100) is much

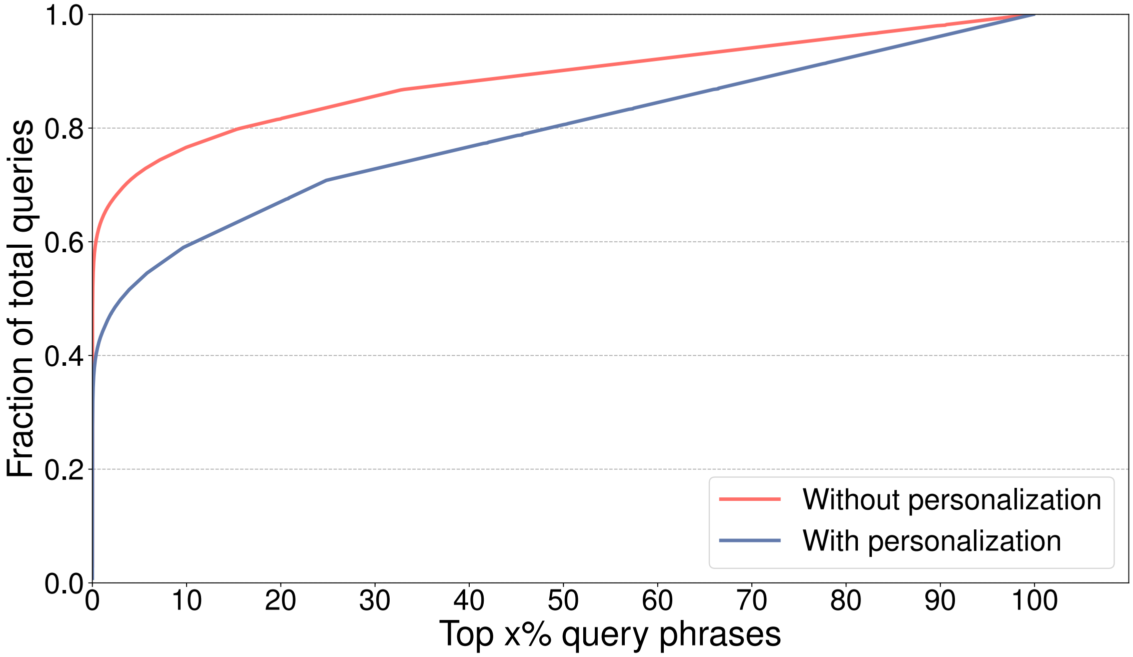


Figure 2.3: Percentage of queries contributed by the top query phrases.

higher than the smallest nonzero average revenue. This shows that the potential revenue of distinct query phrases is highly skewed.

When considering caching for search advertising systems, one important consideration is to avoid potential ad click revenue loss. Since only 1.6% of distinct query phrases have revenue, it seems that it's possible to just not cache any queries with those query phrases. However, since some of those query phrases have very high frequency (a query phrase may have many corresponding queries, but only a few percent of queries end up with ad clicks), these 1.6% of query phrases contribute to 30% of the total queries. Thus we still want to cache these phrases to save the cost but we need a more intelligent refresh strategy to deal with these query phrases with revenue.

Average expected cost-per-click of cached ads

When designing the prediction-based cache, we realize that only predicting the potential revenue of each distinct query phrase is not enough. At each query, we need to predict the revenue loss when using the cached ads list (computed at last refresh) to serve the query. We need additional features that indicate the expected revenue of the cached ads list and the "real" ads list (the potential new ads list if we refresh the cache entry again right now).

For the cached ads list, we choose the average expected cost-per-click (CPC) to indi-

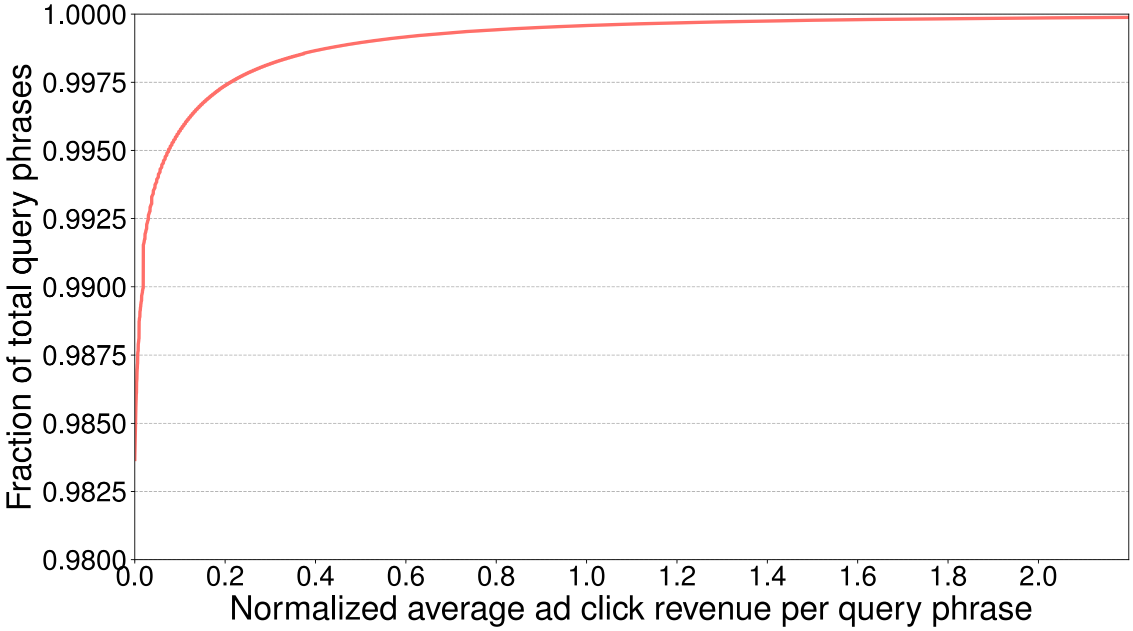


Figure 2.4: CDF of normalized average ad click revenue for each distinct query phrase. The x-axis numbers are normalized by multiplying the same constant coefficient. Note the y-axis starts at 0.98. Outliers (≤ 100) are truncated on the right end of the figure.

cate the expected revenue. The expected cost-per-click for each ad is the expected revenue when the ad is clicked, computed by multiplying the advertiser's bid with the click probability. Then we take the average of all cached ads' expected CPC as the revenue indicator. One thing to note is that the expected CPC is computed conditionally when serving the previous query and the previous user. Thus this average expected CPC is not the exact expected revenue when serving the cached ads to another query. However, it's still a helpful approximation of the expected revenue of the cached ads. Figure 2.5 illustrates the CDF of normalized average expected CPC for the ads selection of each query. More than 55% of queries have nonzero expected CPC, and the distribution is highly skewed.

Incoming query throttling level

When predicting the revenue loss by caching, the "real" ads list for the incoming query is not yet computed since we haven't decided whether refresh the cached ads or not. Thus we use a different signal, the throttling level, as the revenue indicator for the incoming query when we don't use the cache. The throttling level of each incoming query is an abstract integer level that indicates the correlated approximate expected revenue. It is currently used for capacity throttling in Bing Ads so that we could prioritize on serving queries with higher expected revenue when there is a capacity shortage. Unlike CPC which calculates

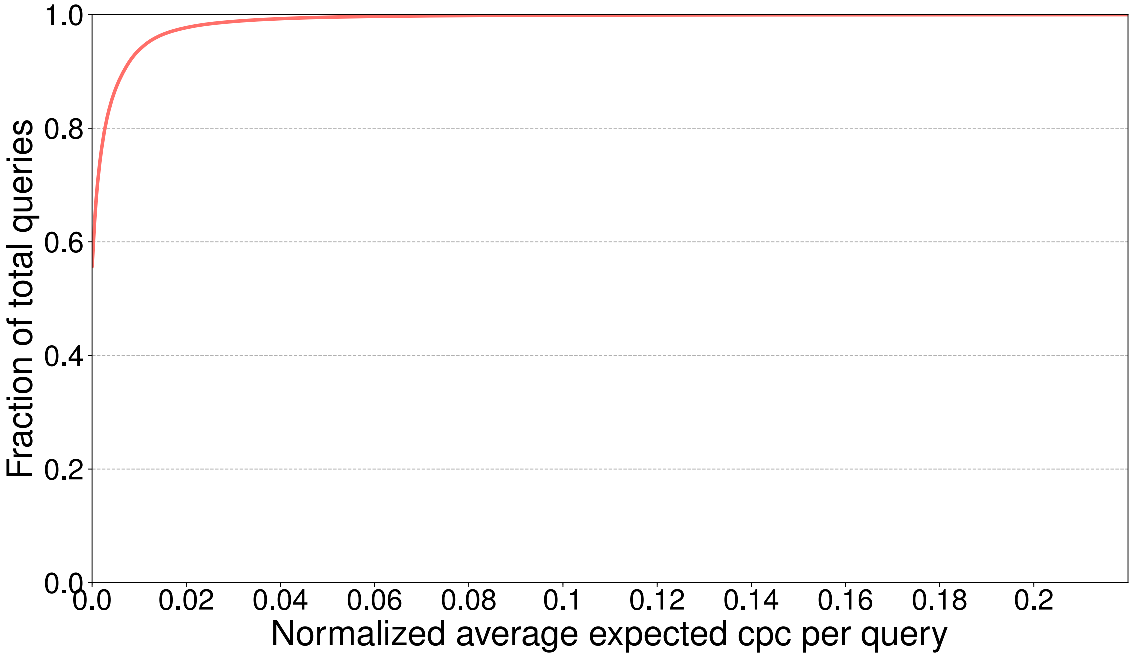


Figure 2.5: CDF of normalized average expected CPC for each query. Normalization uses the same multiplication coefficient as Figure 2.4. Outliers ($\leq$ 329) are truncated on the right end of the figure.

the expected revenue of the selected ads, the throttling level is aggregated from different learning-based revenue predictions using signals from the incoming query (query phrase, phrase category, user information, etc.) and history information such as revenue history. Since throttling level must be computed quickly, it doesn't take the current ad candidates into consideration. However, it's still a helpful approximation of the expected revenue of the incoming query.

Figure 2.6 illustrates the CDF of throttling level for each query. Larger throttling level indicates larger expected revenue. Similar to the average historical revenue and average expected CPC above, the throttling level distribution is also highly skewed: more than 73% of total queries have throttling levels no larger than 20, while the other 27% of total queries have varied throttling levels from 21 to 200.

Summary of the revenue-related features

Analysis above show that the average revenue history, the average CPC of cache entries, and the throttling level of incoming queries are useful features to predict the revenue loss by caching. When designing the heuristic-based cache, we only take the average revenue history into consideration since it is difficult to tune the heuristic with multiple revenue

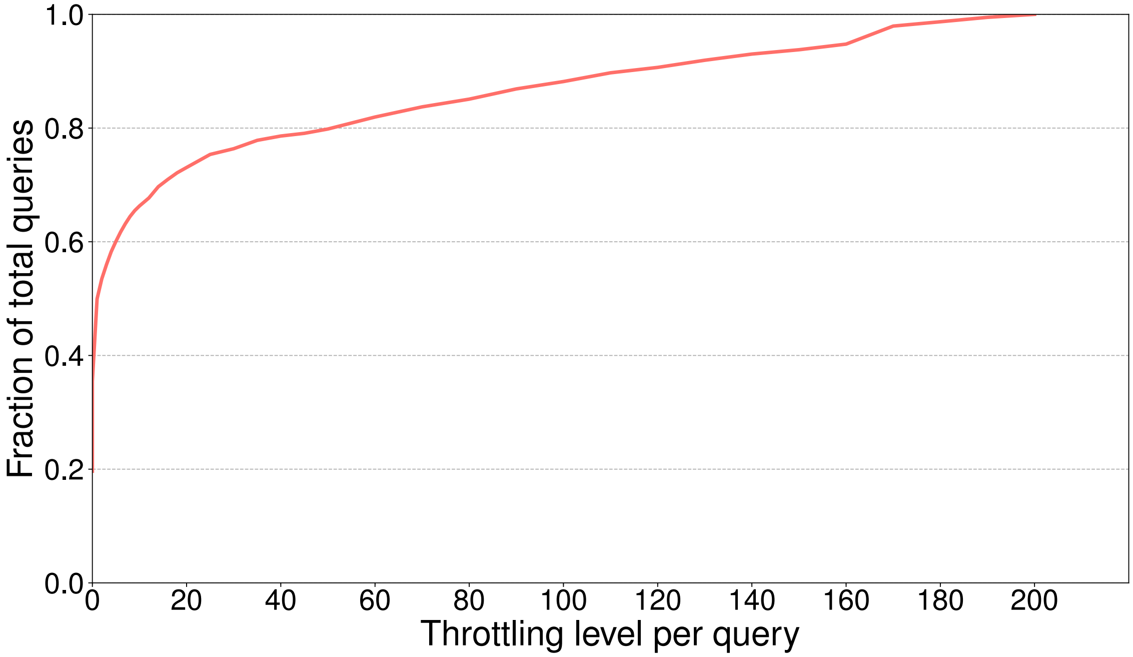


Figure 2.6: CDF of throttling level for each query.

indicators.

In contrast, we incorporate all three features in our prediction-based cache design. If the expected revenue indicated by the throttling level is much higher than the average expected CPC, it means that the expected revenue of the incoming query is much higher than the expected revenue of the current cached ads, and the potential revenue loss could be large as well. By incorporating the average expected CPC and the throttling level features, we are able to compare the expected revenue of the cached ads list and the "real" ads list in the prediction framework, and provide more accurate refresh decisions compared to the heuristics.

2.3.5 Ad-serving cost distribution

Figure 2.7 plots the CDF of the cost indicator for each query. About 40% of the queries have no candidate ads for scoring. There are two main sources of such queries: the corresponding query phrase might be too rare that no advertiser provides ads related to it; or the Bing advertising system recognizes the query as fraud so that it doesn't serve any ad to it. For the other 60% of the queries, each query has thousands of ads in average to be scored. It is still beneficial to cache those queries with zero cost, since it will still save the processing time of the query by skipping the candidate retrieval.

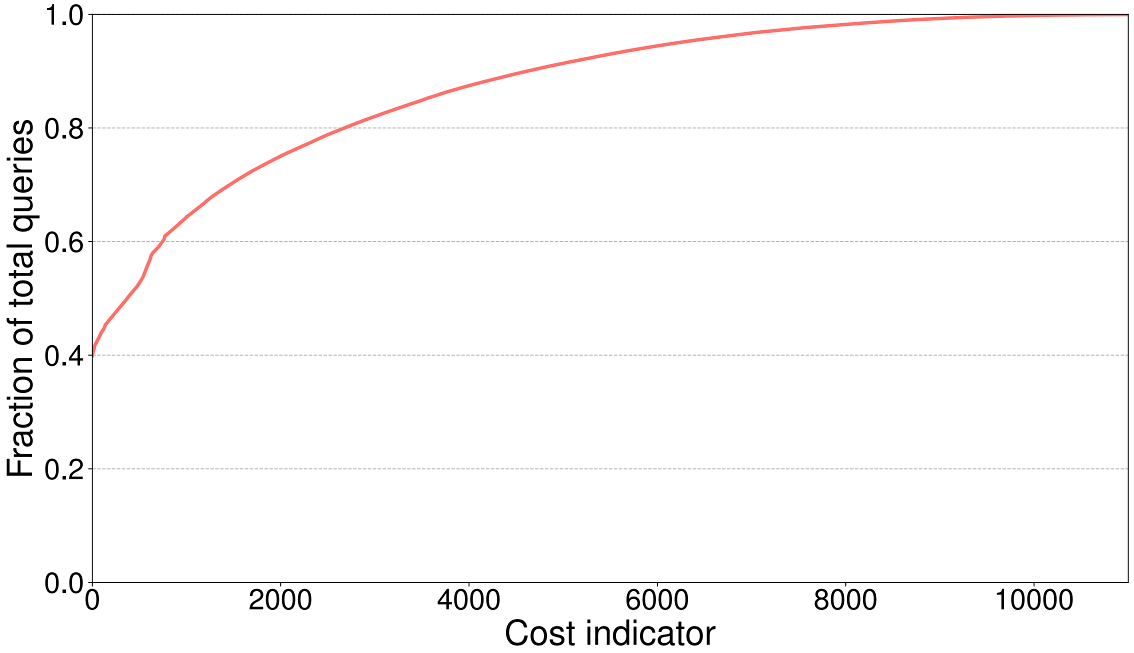


Figure 2.7: CDF of ad-serving cost indicator for each query. A few outliers are truncated on the right end of the figure.

To score thousands of matching candidate ads, it takes tens of milliseconds for hundreds of machines to compute. Based on this cost indicator distribution and the number of dedicated machines, the total machine learning computation cost would typically be around 10% to 30% of the total gross revenue. This cost distribution shows that the learning-based ads selection requires substantial computation power. It'd be preferable to use caching to reduce the number of dedicated machines and the overall cost.

2.3.6 The intrinsic variance of learning algorithm results

Compared to traditional caching, one of the biggest differences for the ad-serving cache is that the cached learning algorithm results (pre-auction ads list) have intrinsic variance. Even for two search queries with the same query phrase, the ads selected by the learning algorithms may vary for three main reasons: (1) Users differ in terms of location, gender, age and so on. This variance affects the decisions of advertisers and publishers; (2) On the advertiser side, ad listings may be removed or added, and the bid budget may change based on different features (time, user location, user gender, etc.); (3) On the publisher side, several different machine learning algorithms are used in parallel for ads selection that use the user and advertiser above as features. Due to this variance of learning algorithm results, it's practically impossible to cache an ads list for a certain query phrase and then

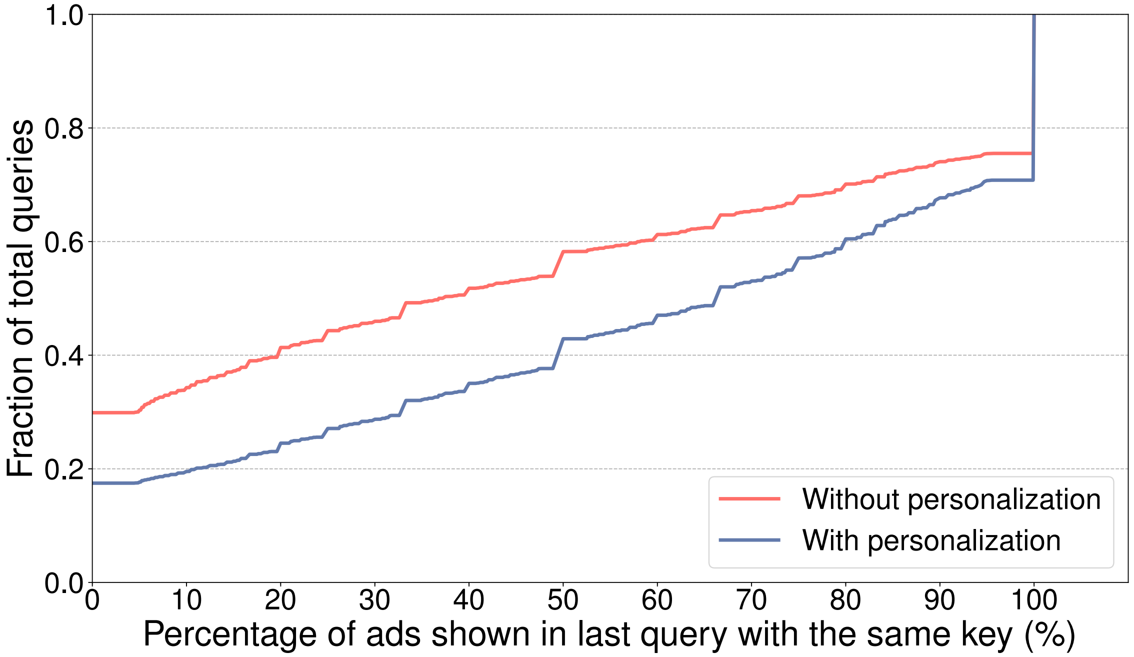


Figure 2.8: CDF of ads list similarity score for each query.

use the cached entry forever without refresh.

To quantify this variance, we calculate the similarity score which is the percentage of ads shown in ads list of last query with the same phrase (when personalization is considered, it has to be both the same phrase and the same three personalization features). Higher similarity score means higher similarity between the two consecutive queries with the same key. Figure 2.8 plots the CDF of this similarity score for each query, both with and without personalization. Note that when calculating the similarity scores: 1) The first query of each query phrase doesn't have the similarity score; 2) When two consecutive queries with the same query phrase both have empty ads list, the second query doesn't have the similarity score.

When personalization is not considered, about 30% of the total queries have 0% similarity. Among these queries, 11% of the total queries belong to the case where the last query with the same query phrase has no ads listed. The other 19% of the queries belong to the case where the non-empty ads list of the last query with the same query phrase has zero intersection with the current ads list. On the other hand, 24% of the total queries have 100% similarity, which means the current ads list can be completely covered by the last ads list related to the same query phrase. Overall, the average similarity is 45%.

When personalization (location, gender, age) is considered, overall the similarity scores increase since the queries with the same query phrase and personalization have more stable ads list. About 17% of the total queries has 0% similarity. Among these queries, 7% of the total queries belong to the case where the last query with the same query phrase has no ads listed. The other 10% of the total queries belong to the case where the non-empty ads list of the last query with the same query phrase has zero intersection with the current ads list. On the other hand, 29% of the total queries have 100% similarity. Overall, the average similarity is 58%.

The similarity score distributions indicate that personalization can increase the similarity score and reduce the variance of the ads list. On the other hand, the similarity score variance shows that the cache needs a dynamic and adaptive refresh policy to keep the average similarity at a high level to avoid revenue loss.

2.3.7 Effect of personalization

As described above, different user information affects the decisions of advertisers and publishers. As a result, queries with the same query phrase and different user information may have different expected revenue. We investigate the queries with one of the top profitable query phrase and the combination of three user features: location, gender, age. For each distinct user info combination, we compute the average revenue of the corresponding queries. This average revenue distribution is highly skewed: more than 94% of distinct user info combinations have no revenue, while the others have varied average revenue. This shows that it could be beneficial to take the user information of both the cached ads list (at last refresh) and the incoming query into consideration.

Using personalization could increase the similarity between ads list, thus reduces potential revenue loss by caching. However, adding personalization info to the cache key increase the number of distinct keys, thus reduce the cache hit rate and cost savings. We consider three personalization features in our workload analysis: location, gender, and age of the user. Adding these three personalization features on all cache keys would double the number of distinct cache entry keys in our workload. In search advertising systems, there are other personalization features (e.g., device type, IP address, and user ID) that could also be included into the cache entry key. In our heuristic-based cache design, we select the three personalization features since they appear to be the key features in Bing Ads system's ads selection, and they are common features that should exist in any search advertising system. In our prediction-based cache design, we also take the user's device type as an additional personalization feature when predicting the potential revenue loss. Exploring how to accommodate more personalization features within limited memory and communication/computation limits is an open direction.

Chapter 3

Caching Strategies for Recommendation Systems

In this chapter, we propose to improve the scoring component for searching advertising systems by caching. Caching systems can save the expensive computation cost by reusing previous scoring results. Effective caching for ads systems is, however, challenging because the ads selected for a previous query may not be those with the best expected revenue for the current query, which could reduce revenue. For example, two users from different states searching for “local furniture store” likely expect different results, and two users with different ages may have different preferences on “classic movies”. Other queries may be invalidated by the progression of time; for example, a product release may cause a shift in expectations for query results for, e.g., “screen pixel”.

The decision about whether a cached result is applicable to a new query depends on both the *key* used to cache it (i.e., does the retrieval key include the same query phrase (and the same user info)?), as well as a determination of whether the previously cached results are still applicable to the new query (i.e., does the cache entry needs a refresh?). In this work, we assume that results are cached based upon the query phrase (optionally combined with personalization features), and use both recency as well as various features to determine whether or not a cached computation result should remain valid.

An ideal refresh policy is revenue-aware: it refreshes only if there will be revenue loss due to serving stale ad suggestions. However, it is hard to predict the revenue loss and make accurate refresh decisions because: (1) There are many involved features from the historical statistics, the incoming query and user, and the cached previous computation result. It is hard to choose which ones and how they jointly affect revenue; (2) The average click-through rate (CTR) is as low as $2 - 3\%$, making it easy to trigger false positive or false negative refresh decisions [1]; (3) The refresh decision must be made quickly, since the whole ads selection process must finish in tens of milliseconds. These challenges make it difficult to build a profitable cache with intelligent refresh policy.

Motivated by the challenges, we first propose an ads cache for search advertising systems employing three domain-specific heuristics to reduce revenue impact and achieve a net profit improvement:

- The *revenue-aware adaptive refresh policy* provides varied refresh decisions based on the potential revenue gain of different query phrases combining historical and instantaneous query information. This enables the cache to incorporate varied refresh strategies for queries with different potential revenue impacts, resulting in both cost savings and low revenue loss.

- The *selective personalization policy* exploits three personalization features (location, gender, age) on only those revenue-sensitive cache entries. This policy makes better use of the personalization features to optimize the cache for both cache hit rate (i.e., cost saving) and revenue impact.

- The *ads list merging technique* combines the ads list from multiple previous computation results of the same query phrase, reducing the likelihood of missing revenue-critical ads.

Although this heuristic-based cache design improves the net profit (compared to the case without cache), there are several limitations in terms of performance and usability. First, the cache considers only 4 features (revenue history and three user features) which is a very small feature set and may reduce the accuracy of refresh decisions. Second, using a refresh frequency to determine whether or not refresh at next few subsequent queries is not sufficient because queries with the same key could still have very different revenue expectations. We need to make dynamic refresh decision at each query. Third, adding personalization to the cache key reduces not only the revenue loss but also the cost savings. Due to the limitations above, the cache sacrifices the total cost savings with many unnecessary cache refreshes in order to achieve a low revenue impact and a net profit improvement. In addition, the incorporated heuristics require nontrivial manual tuning of parameters such as refresh frequencies to achieve better performance. If the workload changes, these parameters must be re-tuned to keep the same performance.

To improve accuracy and eliminate manual tuning, we propose another prediction-based cache design which uses a rapid machine learning algorithm to predict the revenue loss by caching for each query with a richer set of 29 features from the incoming query, the cached entry, and the historical statistics. Based on the gradient boosting regression tree algorithm [35], we build a fast prediction framework that is able to predict whether or not using cached ads would reduce revenue. This prediction framework has rapid prediction time, fast training time, and low storage requirement. Using this prediction framework, we are able to make accurate refresh decision per query and build a prediction-based ads cache that provides better net profit improvement without any parameter tuning.

For the rest of this chapter, Section 3.1 presents the heuristic-based cache design. Section 3.2 presents the prediction-based cache design. Section 3.3 presents the evaluation results. Section 3.4 concludes the caching work.

3.1 A heuristic-based cache design

Domain-agnostic caching mechanisms such as pure LRU or LRU with a fixed refresh rate work poorly for ad serving. Depending on how their parameters are chosen, they either reduce revenue excessively, or fail to reduce the computation cost. In this section, we discuss three domain-specific caching heuristics we devised to avoid these problems.

3.1.1 Ad-serving cache design space

To build an effective ad-serving cache for maximizing cost saving while minimizing revenue loss, we discuss three important design questions as below.

1. What keys to cache? Query phrase is the most common choice as the key. However, since personalized information such as location and gender could affect the ads lists variance, only using query phrase may result in large difference between the cached ads versus the actual ads, causing revenue loss. On the other hand, one may choose to use query phrase together with personalization features as cache key. This approach reduces the potential discrepancies between cached and actual results. However, as studied in web search engine caching, personalization could render lower hit rate since the reuse frequency would be lower — the cost saving of the cache would be less [18]. How to exploit personalization features in the cache design is an important question.

2. What values to cache? The most intuitive answer is to cache the pre-auction ads lists computed by the scoring-based selection from the previous cache miss or last refresh. However, as described in Section 2.3.6, this pre-auction ads list has intrinsic variance, especially when the personalization is not considered. This could be a source of lower similarity scores and higher revenue loss, posing another challenge to the cache design.

3. When to refresh? Due to intrinsic variance of the pre-auction ads list, the ad-serving cache needs an active refresh policy to avoid revenue loss. A basic approach is to have a refresh rate with fixed period or frequency: a higher rate reduces revenue impact but also reduces cost savings, and vice versa. Can we do better? The workload properties discussed in Section 2.3.4 shed some light — the revenue is contributed by only a small portion of distinct query phrases. Could we apply different refresh rate to query phrases with potentially different revenue impact?

Beyond the above three major design questions, cache size and replacement policies are two common aspects to consider during cache design. We found, though, the decision for them at ad-serving cache is fairly intuitive given the properties of the workload.

• With respect to cache size, a small cache could achieve good performance since the query frequency distribution is highly skewed (Section 2.3.3) and the key-value pair we cache has rather small sizes. Thus it's possible to have a large enough cache to store all key-value pairs. Whether the cache size is infinite or not, it is important to actively refresh the cached key-value pairs to keep the freshness of the cached ads list and avoid potential revenue loss.

• Because it is inexpensive to have a large enough cache for frequently accessed items and most of cache update comes from refreshing policy, the choice of the replacement policy becomes less important in this context. We find least recently used (LRU) policy works well, and more advanced policies such as GreedyDual-Size [21] and GD-Wheel [69] only bring marginal benefits.

3.1.2 What keys to cache: selective personalization

To better exploit the benefit of personalization, we propose a selective personalization strategy. For those query phrase with no revenue generated before, we don't consider personalization to save more computation cost. For those query phrases that have revenue history, we use the combination of query phrase and personalization features (location, gender, age of the user) as the key. When a query phrase starts to generate revenue, we insert a new cache entry with the three personalization feature included and remove the old cache entry. By using a subset of key personalization features only for those query phrases that contribute to the total revenue, we could avoid additional revenue loss while minimizing the reduction of cost savings.

3.1.3 What values to cache: ads list merging

To reduce the effect of the ads list variance, we propose to cache a merged ads list based on all previous pre-auction ads lists computed by the scoring-based selection. Specifically, we use a fixed size queue to maintain the cached ads list. The size of the queue is a bit larger than the usual size of a single pre-auction ads list. Whenever a refresh is scheduled, instead of completely replacing the cached ads list, we update the cached ads list by inserting new ads to the head of the queue. If the ad already exists in the queue, we move the ad to the head of the queue. When the queue is full, we evict the ad at the tail of the queue. By caching a merged ads list, we could reduce the variance of the ads list thus avoid revenue loss.

3.1.4 When to refresh: revenue-aware adaptive refresh

Recent works for web search engine caching propose to use hybrid strategies [32, 87] and adaptive refresh frequency [6, 16] based on access frequencies to increase cache hit rate while reducing staleness. For search advertising caching, however, minimizing revenue loss is the primary design goal that other refresh policies do not consider, and this goal requires the refresh policy to take both the revenue history and the staleness (intrinsic variance) of the cached results into consideration.

We propose a revenue-aware adaptive refresh policy which assigns different refresh rates to cache entries based on the revenue history and the ads list similarity score. For those query phrases with no revenue generated before, we assign a fixed and conservative refresh rate to the corresponding cache entries. For those query phrases that have revenue history, the corresponding cache entries have an aggressive and dynamic refresh rate which keeps changing based on the similarity score. After each refresh we compute the similarity score of the old cached ads list based on the new refreshed ads list. We reduce the refresh rate if the similarity score is high, and vice versa. With such adaptive refresh policy, the ad-serving cache could make a better tradeoff between the cost saving and the freshness of the cached ads list. When a query phrase starts to generate revenue, the conservative refresh rate will be replaced by the aggressive refresh rate.

3.1.5 Heuristic-based approach summary

To summarize, we show how the proposed three domain-specific caching heuristics handle cache access and insertion in Algorithm 1 and 2. When handling cache access, we first determine the key based on whether the query phrase has revenue history or not. If so, we combine the query phrase and the personalization information as the key. Otherwise we just use the query phrase as the key. On cache miss, we need to run all ads selection stages and later insert the pre-auction ads list into the cache. On cache hit, we first decide whether or not refresh the cached ads list based on the stored refresh frequency and the frequency counter of the entry. If we do refresh, we first compute the new pre-auction ads list just like a cache miss. If the query phrase has revenue history, we update the aggressive refresh frequency based on the similarity between the cached ads list and the new ads list. Then we merge the new ads list into the cached ads list and reset the frequency counter. If we don't refresh, we increment the frequency counter. Finally we move the entry to the head of the LRU queue, and return the cached ads list. If later any ad is clicked by the user, we will add the query phrase to the list of query phrases with revenue history.

Algorithm 1: How the proposed heuristics process cache access

```
input : Query phrase: $q$,
        personalization information: $p$,
        list of query phrases with revenue history: $R$,
        the hashtable that stores the cache entries: $H$,
        the LRU queue: $L$.
output: Ads list on cache hit, or $null$ on cache miss.
$key \leftarrow q$
if $q \in R$ then
    $key \leftarrow key + p$
end
if $H[key] =$ null then
    return $null$
end
freq $\leftarrow H[key]$.refreshFreq
cnt $\leftarrow H[key]$.refreshCnt
if cnt%freq = 0 then
    // do refresh
    $\{ad\} \leftarrow$ new pre-auction ads list computed by scoring-based selection
    if $q \in R$ then
        similarity $\leftarrow$ compare($H[key]$.cachedAds, $\{ad\}$)
        update $H[key]$.refreshFreq based on similarity
    end
    $H[key]$.cachedAds $\leftarrow$ merge($H[key]$.cachedAds, $\{ad\}$)
    $H[key]$.refreshCnt $\leftarrow 1$
else
    increment $H[key]$.refreshCnt
end
move $H[key]$.lruNode to the head of $L$
return $H[key]$.cachedAds
(if there is any ad get clicked, insert $q$ into $R$.)
```

When handling cache insertion, we first determine the key based on whether the query phrase has revenue history or not. If the cache is full, we evict the least recently accessed cache entry. Then we cache the pre-auction ads list computed by the scoring-based selection, set the frequency counter, add the entry to the head of the LRU queue, and set the refresh frequency based on the query phrase's revenue history.

Algorithm 2: How the proposed heuristics process cache insertion (after Algorithm 1 returns null)

```
input: Query phrase: $q$,
       personalization information: $p$,
       pre-auction ads list to be cached: $\{ad\}$,
       list of query phrases with revenue history: $R$,
       the hashtable that stores the cache entries: $H$,
       the LRU queue: $L$.
$key \leftarrow q$
if $q \in R$ then
    delete $H[key]$ if exists
    $key \leftarrow key + p$
end
while $size(H) \geq$ cache size do
    // eviction
    evict the tail of $L$ and delete its entry in $H$
end
$H[key]$.cachedAds $\leftarrow \{ad\}$
$H[key]$.refreshCnt $\leftarrow 1$
$H[key]$.lruNode $\leftarrow$ new node at the head of $L$
if $q \in R$ then
    $H[key]$.refreshFreq $\leftarrow$ aggressive frequency
else
    $H[key]$.refreshFreq $\leftarrow$ conservative frequency
end
```

3.2 A prediction-based cache design

As discussed at the beginning of this chapter, the heuristic-based cache design has several drawbacks in terms of performance and usability. To improve accuracy and eliminate manual tuning, we develop another prediction-based cache design. This section presents the requirements, features, and empirical evaluations of the prediction framework.

We use the gradient boosting regression tree algorithm [35] to build a rapid, accurate and flexible prediction framework that predicts the potential revenue loss by caching. Gradient boosting decision trees are an ensemble model of decision trees. A decision tree is a flowchart-like tree in which each internal node represents a “test” on a feature (e.g., whether the feature value is larger than 10), each branch represents the outcome of the test, and each leaf node represents a prediction value. The gradient boosting decision tree model trains a set of weak decision tree models in an iterative fashion. At each training iteration, a new weak decision tree is trained attempting to improve from the previous tree. At inference, the prediction is computed as the weighted sum of the predictions from all weak models. As a result the number of training iterations affects both the model size and the prediction accuracy/latency.

We selected this decision tree approach because of several of its strengths: Both training and inference are fast, and the models allow for introspection: decision tree models allow us to identify the importance of individual features by the total error reduction per split in the tree, which is helpful during feature exploration and for explaining why the system works. For the same reasons we also use gradient boosting decision trees when predicting the search termination condition for approximate nearest neighbor search.

3.2.1 Requirements

We desire three attributes in the prediction framework: accuracy, prediction overhead, and flexibility. We use the standard metrics of prediction, namely precision ($|A \cap P|/|P|$) and recall ($|A \cap P|/|A|$), where A is the set of true queries with revenue loss, and P is the set of predicted queries with revenue loss when using the cache ads.

Accuracy Given an incoming query and the corresponding cached ads, we want the prediction framework to predict whether using the cached ads to serve the query would lead to revenue loss or not. The difficulties of this prediction problem come from two aspects: (1) Biased sample set. Only about 3% of queries have ad click revenue [1]. (2) Intrinsic uncertainty of user click behavior. In fact, for the same pair of query and ads, different users may have quite different click behaviors. Even for the same user who searches the same query multiple times, they may either click or not click on the same ads. This large intrinsic uncertainty means that the upper bound of classification accuracy would be low. Since the average revenue per click is much higher than the average computation cost per query, a false negative prediction is much more harmful to net profit than a false positive prediction. Thus recall is a more important metric than precision in this problem.

Prediction overhead The latency overhead involved in performing prediction must be small to keep the interactive nature of web search. Prediction itself adds additional work to the advertising system, since we will still perform the machine learning-based ads selection if the prediction shows that using cached result lead to revenue loss. Since the ads serving execution takes tens of milliseconds, we set the latency budget of the prediction framework at less than one millisecond.

Flexibility Since we have different requirements on precision and recall, the ability to adjust the threshold of defining queries with nontrivial revenue loss allows the predictor to adapt to varying workload and different performance requirements. We thus abstract the prediction as a regression problem (of estimating the revenue loss by caching) rather than a classification problem (of deciding whether serving the query with cached ads lead to nontrivial revenue loss or not).

3.2.2 Features

Arriving queries have features from the incoming query and user, from the cached entry based on the cache entry key, and from the historical statistics based on the query phrase. In this section, we describe the features that can be used for prediction and analyze the importance of the features.

Space of features

We investigate 29 features that meaningfully correlate with the potential revenue loss by caching, which we categorize into incoming query features, cached entry features, and statistic features as listed in Table 3.1. To keep the prediction framework fast enough and limit the storage overhead, we select a subset of features that commonly exist in any search advertising system and have high impact on ads selection in Bing Ads.

Incoming query features Incoming query features describe the incoming query and user. We use the throttling level described in Section 2.3.4 to represent the potential revenue of the incoming query, and use an internal query category id to distinguish different query phrases. We select location, gender, age, and device type to describe the incoming user. The location includes features at 5 levels, where level 1 denotes country and higher levels denote smaller geographical regions. Since these features come with the incoming query, they do not incur storage overhead.

Cached entry features Cached entry features describe the cached ads and the previous user at last refresh of ads selection result. We use the average expected CPC described early on to represent the expected revenue of the cached ads, and we use the cost indicator to describe the number of matching ad listings at last computation. Since the cached ads were selected for a different user, we use the same set of features to describe the user at the previous computation. We use two additional features to describe when was the last refresh of the cached ads. These features require storage overhead for each cache entry.

Feature	Description	Storage Overhead
ThrottleLevel	Throttling level of incoming query	None
QueryCategory	Category id of incoming query	None
Location1-5	Location of incoming query's user (5 levels, 1 denotes country)	None
Gender	Gender of incoming query's user	None
Age	Age group of incoming query's user	None
DeviceType	Device type of incoming query's user	None
AvgCPC	Average expected CPC of cached ads	Per cache entry
Cost	Cost indicator of last refresh	Per cache entry
CLocation1-5	Location of cached ads' user (5 levels, 1 denotes country)	Per cache entry
CGender	Gender of cached ads' user	Per cache entry
CAge	Age group of cached ads' user	Per cache entry
CDeviceType	Device type of cached ads' user	Per cache entry
LastRefDur	Duration gap between last refresh and incoming query	Per cache entry
LastRefFreq	Occurrence gap between last refresh and incoming query	Per cache entry
AvgRev	Average revenue per query	Per query phrase w/ revenue
ClickPeriod	Click period	Per query phrase w/ revenue
ClickFreq	Click frequency	Per query phrase w/ revenue
RefPeriod	Refresh period	Per query phrase
RefFreq	Refresh frequency	Per query phrase
AvgLossDur	Duration-based average loss rate	Per query phrase
AvgLossFreq	Occurrence-based average loss rate	Per query phrase

Table 3.1: Space of the features.

Statistic features Statistic features describe the historical statistics for each query phrase. We use the average revenue per query and click period/frequency to represent the revenue history. We also use refresh period/frequency to describe refresh history. In addition, we use two average loss rate numbers to represent the changing rate of ads selection over time. For two ads lists A and B from two consecutive refreshes, the loss rate is computed by $(1-|A \cap B|/|A \cup B|)/(\text{duration or occurrence gap between two refreshes})$. All the statistic features above are aggregated using only the training data, not including the whole history log and the test data. On the other hand, we keep updating these statistics using the *past* query information during the cache simulation. These features require storage overhead for each distinct query phrase.

Feature	Importance	Feature	Importance
AvgRev	1	CLocation5	0.02773
AvgCPC	0.28498	Location5	0.02318
ThrottleLevel	0.19877	QueryCategory	0.02148
Gender	0.07363	AvgLossFreq	0.01925
DeviceType	0.05106	AvgLossDur	0.01558
RefPeriod	0.05100	Location4	0.01313
ClickPeriod	0.03050	Age	0.00995
ClickFreq	0.02877		

Table 3.2: Top-15 features ranked by the importance obtained from boosted regression tree.

Feature analysis

To study which features are good predictors of revenue loss by caching, we use per-feature gain from boosted regression tree where the importance of a feature is proportional to the total error reduction per split in the tree. Table 3.2 shows the 15 most important features, with each importance normalized to the highest value. We observe that the top 3 features are all directly related to revenue history or expected revenue. Although the throttling level is the only feature that has the knowledge of all revenue history, it is only the 3rd most important feature in our experiments. This is because throttling level is designed only to predict the approximate expected revenue level (with only 200 levels), not the precise expected revenue. The next two top features are gender and device type of the incoming user, as female users and PC users tend to have higher expected revenue in some traffic. The highest level location of both incoming user and previous are also helpful since local advertisers tend to spend bid budget at only neighboring regions.

3.2.3 Empirical evaluation

To find the ground truth revenue loss for training data built from the history log, we need to estimate the revenue loss when serving a different cached ads list to a query. We estimate this revenue loss by aggregating the revenue of ads clicked by the user in history that are not included in the cached ads list. Although there exist more accurate revenue loss estimations such as the auction simulation we use in Section 3.3, such calculations take too long to compute thus are not fit for fast model training. Thus we use a fast yet reasonable calculation to build the training data.

To build the training data, we simulate an infinite-size cache over the history log. For the first query of each distinct key (query phrase with optional personalization informa-

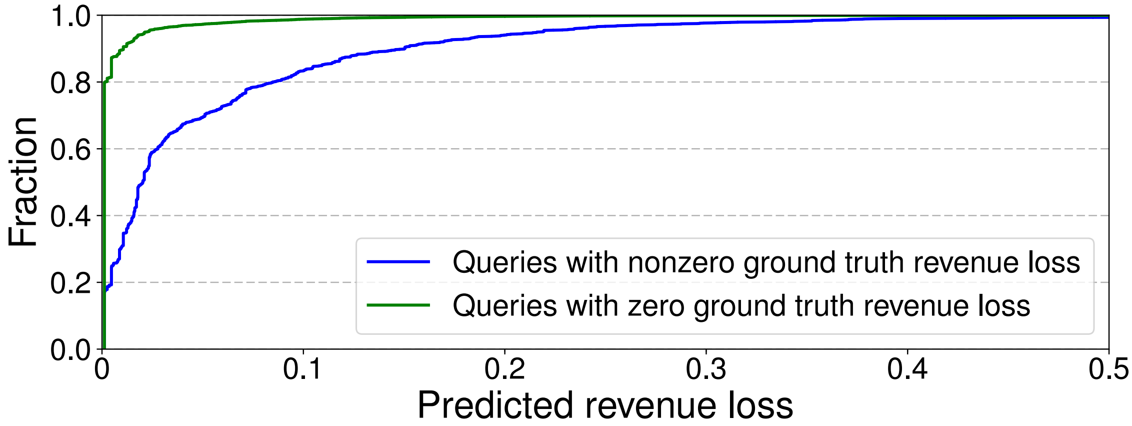


Figure 3.1: CDF of predicted revenue loss for queries with zero and nonzero ground truth revenue loss.

tion), we insert the selected ads into the cache. For each following query, we first compute the ground truth revenue loss based on the click history and the cached ads, and generate the training entry row. Then we refresh the cached ads if the ground truth revenue loss is positive or at the 20th cache hits as a conservative mandatory refresh. As a result we get one row of training data per query and the whole training data includes both zero and nonzero ground truth revenue loss. Despite the overall click-through rate being only $2-3\%$, we choose not to re-balance the training data since 1) there are still hundreds of thousands of clicks in a hour; and 2) we want to keep the low-CTR pattern in the training data just as the real workload.

We train the prediction framework using the Light Gradient Boosting Machine (LightGBM) framework [62]. After training the regression model, we need to determine the threshold that distinguish the queries with nontrivial revenue loss by caching so that the prediction framework can decide when to refresh the cache entry. To do so we compare the ground truth and predicted revenue loss of the training data (or test data based on other log). As plotted in Figure 3.1, we find that for any feature sets we test, there always exists a threshold value (close to the minimum prediction value) where most (e.g., 80%) of the queries with zero ground truth revenue loss have predicted revenue loss less than the threshold, and vice versa. Thus we are able to find the threshold by a simple binary search. Using any threshold with lower value would greatly harm the accuracy of the prediction framework. On the other hand, a slightly higher threshold has marginal effect on the accuracy since the remaining queries, with either zero or nonzero ground truth revenue loss, have predicted revenue loss much higher than the threshold (i.e., it's very difficult to distinguish those remaining queries by the prediction framework with any threshold).

We collect 3 hour log to get hundreds of millions of training entries, and perform 5-

	Precision (%)	Recall (%)
All features	5.02	83.82
Top 15 features	4.80	82.26
Top 3 features	4.03	74.39
Heuristic-based	0.52	51.47

Table 3.3: Prediction accuracy of prediction framework with different feature sets, comparing with the heuristic-based approach.

fold cross validation with 5 repetitions to avoid biased results. We compare the accuracy of the prediction framework with the heuristic-based approach. For the heuristic-based approach, we use the last one day log before the training to build the list of query phrases with nonzero revenue history, and use aggressive refresh frequencies and personalized cache keys for those phrases. Using more history log to build the list would improve the accuracy of the refresh heuristic. However, based on our experiments using more than one day log only marginally improves accuracy due to the diminishing returns on number of frequent query phrases with nonzero revenue.

Table 3.3 presents the average of precision and recall for the prediction framework with different feature sets, comparing with the accuracy of the heuristic-based approach. As expected, the precision is much lower than the recall. This is because (1) we have different requirements on the two metrics, and (2) it is very hard to make a perfect prediction due to the low click-through rate and huge intrinsic uncertainty in user behavior. On the other hand, low precision is acceptable since false positive errors have no/small effects on the gross revenue/net profit, respectively. For the prediction framework, using the top 15 features has similar accuracy to using all features. On the other hand, using only the top 3 features reduces the accuracy but still outperforms the refresh heuristic, since the refresh heuristic only use average revenue history and three incoming user information as features.

Table 3.4 compares the training time, average prediction overhead per query, and storage overhead per one million distinct query phrases when using different feature sets. All the numbers are measured when running the prediction framework on a single machine with Intel Xeon E5-2680 v2. As expected, using fewer features reduces all three metrics. On the other hand, all approaches achieve rapid prediction with acceptable storage requirements.

3.3 Evaluation

To evaluate the proposed heuristic and prediction-based cache designs, we build a cache simulator based on the production logs of Bing advertising system. Each timestamped log

	Training time (s)	Avg prediction overhead (μs)	Storage overhead per 1 million phrases (GB)
All features	71	170	43.9
Top 15 features	50	154	25.7
Top 3 features	17	122	5.7

Table 3.4: Training time, average prediction overhead, and storage overhead of the prediction framework. The average prediction overhead is measured using the Python API. When using the C++ API (as we do in the adaptive ANN search termination condition work), the prediction overhead would be even lower.

entry represents the information related to a single search query: the query phrase, the personalization features (location, gender, age and device type of the user), cost indicator, pre-auction ads list (output of scoring-based selection and the ads list we want to cache as well), which ads got clicked and the corresponding revenues. The cache simulator reads log entries chronologically, makes caching decisions (insertion, eviction, refresh) based on the heuristics or the prediction, and evaluates the caching performances. We simulate the logs on Wed Jun 7th 2017 which is the same as what we analyzed in the Section 2.3. We simulate a cache with LRU replacement policy and one million entries, which is large enough to cache the top query phrases. Since the value of each cache entry stores an ads list with variable numbers of ads, cache entries may have different sizes. However, each ad in the list only takes a few kilobytes including the corresponding metadata.

Section 3.3.1 describes the implementation of evaluated cache designs. Section 3.3.2 describes the performance metrics we use. Section 3.3.3 presents the main results comparing the performance of different cache designs. Section 3.3.4 provides post analysis.

3.3.1 Implementation of cache designs

We implement and evaluate three different cache designs. First we build a domain-agnostic traditional cache with a fixed-rate refresh so that each cache entry will refresh at 5th cache hits as a moderate refresh rate. (We test different refresh rates and all cases end up with worse performance and net profit impact than the proposed cache designs.) This cache doesn't consider the wall clock time when making refresh decisions because our experiments show that adding consideration of wall clock time doesn't have significant effect on revenue impact reduction. At last, this cache doesn't consider any personalization information and the key of each cache entry is the query phrase itself.

Then we build the heuristic-based cache which incorporate three domain-specific caching mechanisms. First, the revenue-aware adaptive refresh policy assigns different refresh frequencies based on the revenue history of the previous day. Cache entries without revenue history have a fixed refresh frequency of 20; cache entries with revenue history have a dynamic refresh frequency that starts from 1 (always refresh) and then increments/decrements based on the similarity between the cached ads list and refreshed ads list. If the similarity is more than 90%, we increment the refresh frequency by 1. If the similarity is less than 70%, we decrement the refresh frequency by 1 (or unchanged if the frequency is already 1). Second, the selective personalization policy combines query phrase and personalization information as the cache entry key for the query phrases with revenue history. Third, the ads list merging technique combines the ads list from multiple previous computation results of the same cache key. We use a fixed size queue to maintain the merged cached ads list. The size of the queue is larger than but at the same magnitude of the usual size of a single pre-auction ads list. Thus the increased size of pre-auction ads list won't affect the processing time of the final auction process.

Finally we build the prediction-based cache. We use 3 hours of logs (19-22PM) on Jun 6th (the day before cache simulation) to build the training data and train the prediction framework using the LightGBM library [62]. We use the next one hour log (22-23PM) as validation data to adjust the threshold as described in Section 3.2.3. Using longer training data would improve the accuracy of the prediction framework. We decide to train with 3 hours log because (1) we want to show that using a relatively short period of training data is already enough to train an accurate and stable model for several days, and (2) our experiments show that using more training data has marginal caching performance improvements for our cache simulations.

We use this trained prediction framework to predict the revenue loss by caching during caching simulation at each cache hit. If the predicted revenue loss is lower than the threshold, we serve the cached ads without refresh. We use only query phrase as cache entry key since it maximizes the performance of prediction-based cache. More precisely, our experiments show that adding personalization of cache keys to the proposed cache doesn't help much on avoiding additional revenue impact but greatly reduces the total cost savings. This is because the prediction framework already takes the personalization features into consideration when marking refresh decisions.

3.3.2 Performance metrics

To compare the performance of different caching design, we use four performance metrics as below.

1. Hit rate Hit rate is one of the basic caching performance metrics. When a refresh is triggered at cache hit, we count it as a cache miss since the refresh requires the candidate selection and scoring-based selection to update the cached ads list.

2. Percentage of cost saving We calculate the caching cost saving by total accumulated cost indicator on cache hits divided by total accumulated cost indicator over all queries. The higher cost saving the better. For the prediction-based cache, we also need to take the cost of prediction framework into consideration. Since each ads selection on cache miss takes tens of milliseconds (on hundreds of machines) and each prediction takes less than 200 microseconds (on a single machine), we estimate that the total prediction cost should be no more than 1% of total ads selection cost when there is no cache. Thus we always subtract the cost saving by 1% for the prediction-based cache case. This also shows that using perdition-based cache design doesn't add significant cost to the caching system compared to the heuristic-based design.

3. Revenue impact It is impossible to exactly calculate the revenue impact of caching in simulations, since we don't know user's action when the presented ads are changed. To scientifically estimate the revenue impact of caching, we use an offline auction simulator available in Bing Ads to simulate the whole auction process. When we use a cached ads list to serve a query, the auction simulator uses a click prediction framework to recalculate the click probability for each cached ad based on the current query and user. When a cached ads list computed for a user is served to another user with different features such as gender and age, the predicted click probability will drop.

As an alternative calculation, we also consider a pessimistic revenue impact. This calculation only count the ad click revenue if the clicked ads are cached on cache hits. This is a pessimistic estimation since the user may click other ads even though the actual clicked ads are not presented. We use this deterministic revenue impact calculation as a post analysis in Section 3.3.4.

4. Net profit impact The net profit equals the total revenue subtracted by the total cost. As mentioned in Section 2.3, 0.1 to 0.3 would be a representative range of learning cost-to-revenue ratio for search advertising systems. Since there exists other operation cost for Bing Ads, we present the net profit impact as absolute values: The total revenue of Bing Ads for fiscal year 2016 4th quarter is $1465.85 million based on Microsoft earnings release [2]. Thus we calculate the expected net profit impact for the quarter as (total revenue × pessimistic revenue impact) + (total revenue × cost-to-revenue ratio × cost saving).

3.3.3 Comparing different cache designs

We evaluate the three cache designs based on the cache simulations and the auction simulations. Figure 3.2 illustrates the hit rates, cost savings, revenue impacts, and Figure 3.3

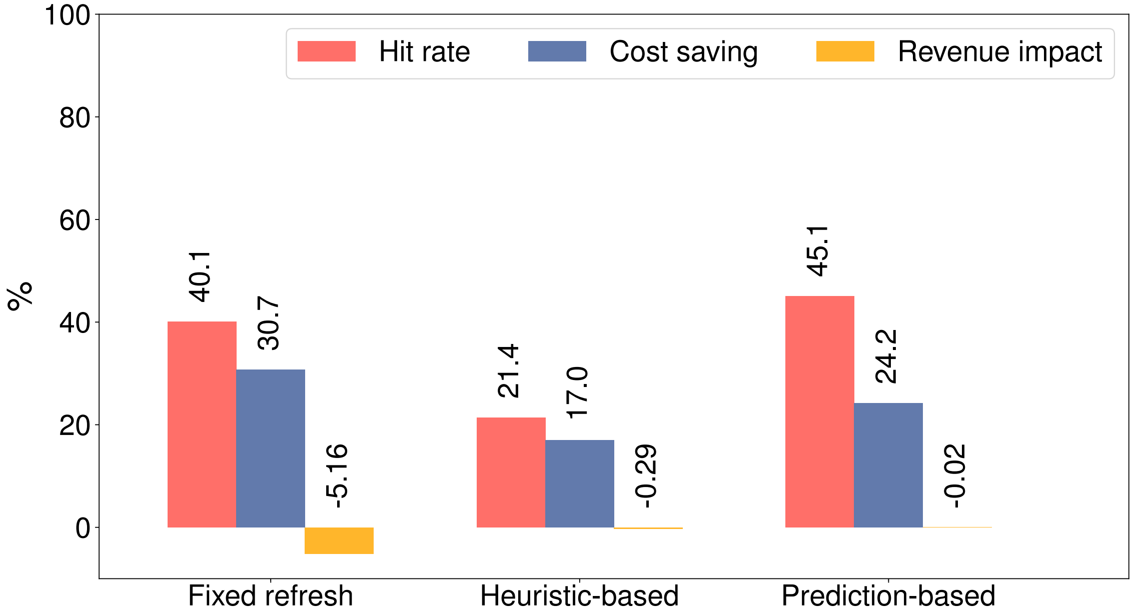


Figure 3.2: Hit rate, cost saving, and revenue impact for naive fixed refresh rate, heuristic-based cache, and proposed prediction-based cache. For all the numbers the higher the better.

illustrates the net profit impacts for the three cache designs. The cache with fixed rate refresh has high hit rate (40.1%) and high cost saving (30.7%), but the revenue impact is as bad as -5.16% since no revenue information is considered at caching. As a result, this cache leads to a net profit impact of -30.6 to 59.4 million dollar for Bing Ads in FY16 Q4 (based on 0.1 to 0.3 cost-to-revenue ratio).

The heuristic-based cache greatly improves the revenue impact from -5.16% to -0.29%. On the other hand, both the hit rate (21.4%) and cost saving (17.0%) are dropped since the cache uses a very aggressive refresh strategy to avoid revenue impact. Eventually this cache has a net profit impact of 20.7 to 70.5 million dollar.

Compared to the heuristic-based cache design, the prediction-based cache is able to achieve a better revenue impact (-0.02%) with much higher hit rate (45.1%) and cost saving (24.2%). This is because the prediction framework is able to accurately predict the revenue loss by caching. Overall the prediction-based cache uses 87% less refreshes compared to the heuristic-based cache. For those query phrases that don't have revenue history in last day, the prediction-based cache makes much less false negative errors (no refresh at revenue loss) since it takes additional features such as throttling level and personalization features into considerations. For those query phrases with revenue history, the prediction-based cache makes much less false positive errors (refresh at no revenue loss) since it doesn't simply apply an aggressive refresh frequency, but makes separate refresh

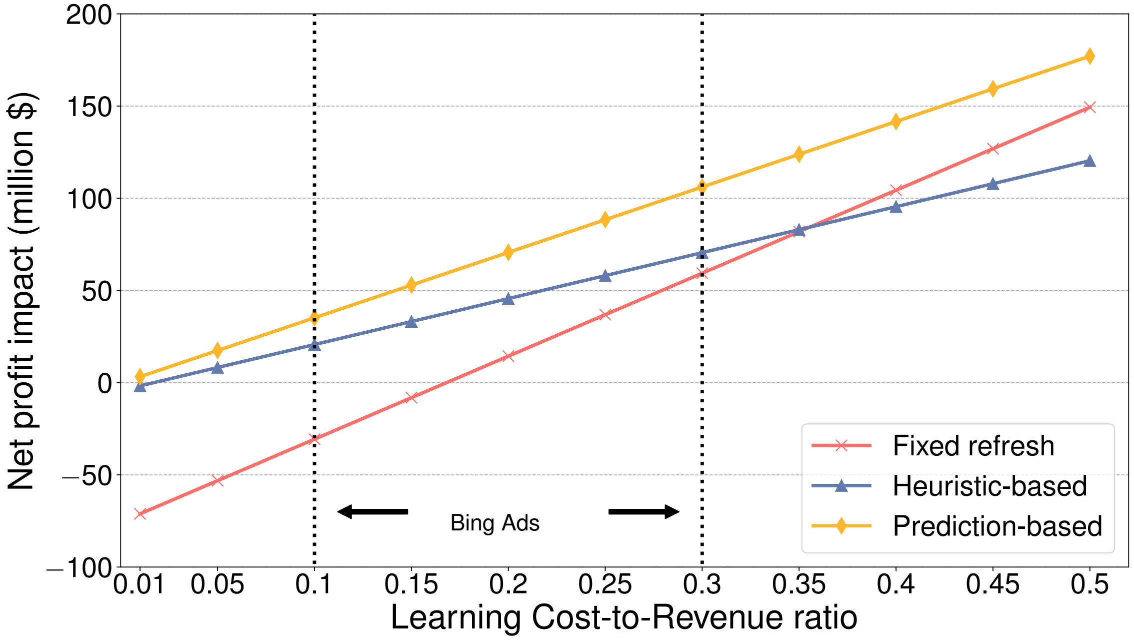


Figure 3.3: Net profit impact at different cost-to-revenue ratios (0.1 to 0.3 as representative range). When there is no cache the profit impact is zero.

decision at every query. Eventually the prediction-based cache is able to greatly reduce revenue impact with less cache refreshes and no personalization for the cache entry keys.

Compared to the cache with fixed refresh rate, the prediction-based cache has higher hit rate but lower cost saving. This is because the prediction framework predicts that queries with higher cost on cache miss tend to have higher revenue loss expectation. Our proposed cache design achieves the best net profit impact of 35.2 to 106.1 million dollar. In addition to the representative cost-to-revenue ratio range between 0.1 to 0.3, we also plot in Figure 3.3 the cases for even smaller or larger cost-to-revenue ratios. The traditional cache provides better net profit at higher cost-to-revenue ratio since it has the most cost savings. However, the prediction-based cache dominates the net profit improvement at any ratio.

3.3.4 Post analysis

Alternative revenue impact calculation

The offline auction simulator we use for the revenue impact calculation is not a public tool. Different revenue impact prediction algorithms may produce different numbers. Thus we also consider an alternative pessimistic revenue impact calculation as described in Section 3.3.2. Under this calculation, the traditional cache with fixed refresh has a revenue impact as bad as -15.2%. The heuristic-based cache has a revenue impact of -2.5%, which is similar to the number reported in the previous work. On the other hand, the proposed prediction-based cache has a revenue impact of -1.0%. Using this pessimistic revenue impact calculation leads to higher revenue impact for all approaches. But the prediction-based cache is still able to provide the least revenue impact.

Heuristic-based cache: parameter tuning

For the heuristic-based cache design, the proposed adaptive refresh policy depends on 4 parameters: the fixed refresh frequency for entries without revenue; the initial refresh frequency for entries with revenue; and the low/high watermark for changing the aggressive frequency based on the similarity score. Changing the fixed refresh frequency for entries without revenue mostly just affect the cost saving. Changing the initial refresh frequency for entries with revenue has noticeable affect on revenue loss. This is because the intrinsic variance of ads lists makes it necessary to always refresh for some of revenue-sensitive phrases. Similarly, using stricter low/high watermark increases number of refreshes, and reduces both revenue loss and cost saving. Manual tuning of these parameters is necessary to maximize the heuristic-based cache performance.

Prediction-based cache: comparing different feature sets

We evaluate the prediction-based cache with different feature sets as illustrated in Figure 3.4. Using top 15 features has nearly the same performance compared to using all features. However, using only the revenue-related top 3 features lead to a slightly worse performance. This shows that other top features such as information of the incoming user and the previous user on refresh are still beneficial to caching performance. On the other hand, all the three cases outperform the heuristic-based cache design.

Prediction-based cache: temporal stability of the prediction

To evaluate the temporal stability of the prediction framework, we use 3 hour log from three different days before cache simulation to build the training data. As illustrated in Figure 3.5, using training data from different days doesn't have much different caching performance. This shows that a prediction framework built from 3 hour log is able to accurately predict the revenue loss in at least next 3 days. This also implies that the proposed features in our framework are representative to capture the stable patterns in terms of revenue expectations.

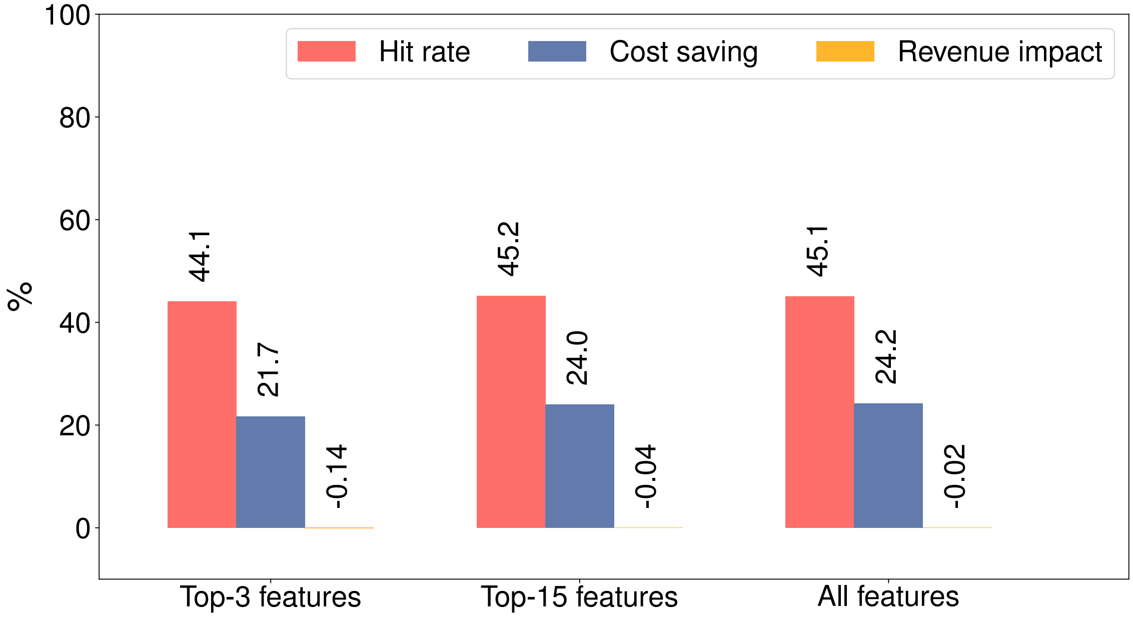


Figure 3.4: Hit rate, cost saving, and revenue impact for prediction-based cache with different feature sets. The corresponding net profit impacts are: \$29.8 to \$93.4 million, \$34.6 to \$105.0 million, and \$35.2 to \$106.1 million.

3.4 Conclusion

Complex machine learning algorithms enable search advertising systems to select the best ads from a large candidate pool thus improve the total gross revenues. On the other hand, these expensive computations bring huge operation cost for all traffics regardless of the revenue expectations. The highly skewed frequency distribution of search queries provides opportunities for caching the learning computation results. However, as we learn from workload analysis of the Bing advertising system, the intrinsic variance of the learning algorithm results leads to substantial revenue loss for traditional domain-agnostic and revenue-agnostic cache designs. A traditional cache with fixed refresh rate can reduce cost by up to 30.7% while having negative revenue impact as bad as -5.16%, and a net profit impact between $\$-30.6$ and $\$59.4$ million.

We propose two cache designs to balance the ads quality and computation cost: one heuristic-based and one prediction-based. The heuristic-based cache is able to provide net profit gain, but the limited feature selection with nontrivial tuning lead to a strategy that sacrifices the cost savings to avoid revenue loss. In contrast, the prediction-based cache accurately predicts the revenue loss by caching, and use it to guide cache refresh decisions. Compared to the heuristic-based cache, the prediction-based cache is able to improve the cost saving (from 17% to 24%), the revenue impact (from -0.29% to -0.02%), and the net

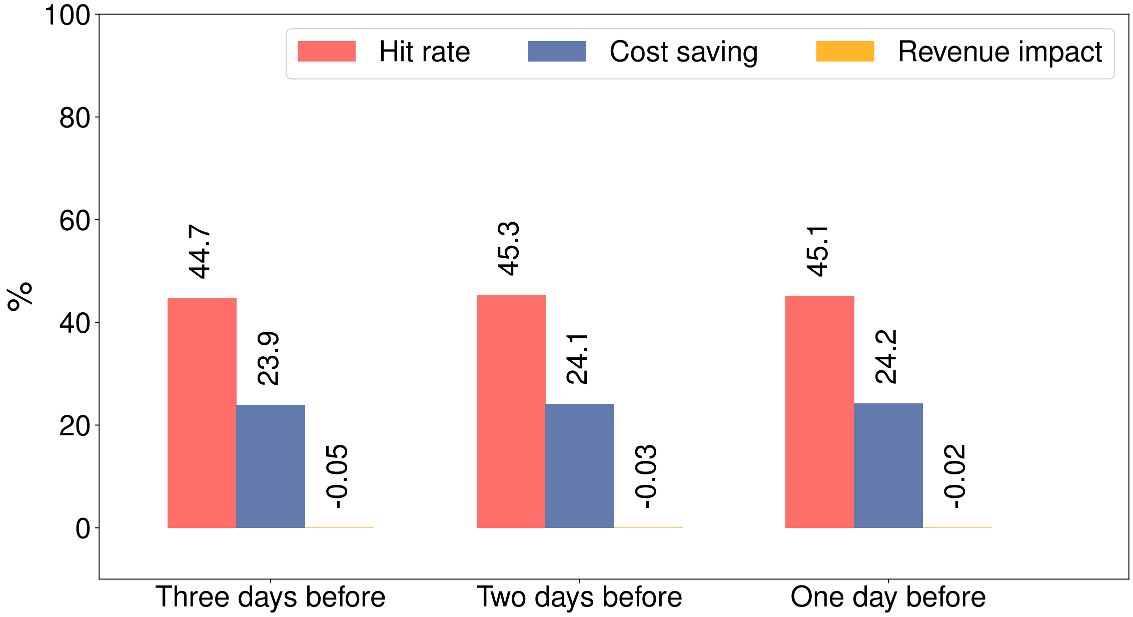


Figure 3.5: Hit rate, cost saving, and revenue impact for prediction-based cache with different training data. The corresponding net profit impacts are: $34.3 to $104.2 million, $34.9 to $105.5 million, and $35.2 to $106.1 million.

profit impact (from [$20.7, $70.5] million to [$35.2, $106.1] million) based on simulation results on traces from Bing Ads.

Our work reassures the advantages of using fast machine learning algorithms instead of manually-tuned heuristics to solve performance tradeoff questions in the search advertising context, and recommendation systems in general. These machine learning techniques enable us to incorporate a rich selection of features, measure the importance of each feature, and use the top features to make fast and accurate decisions without parameter tuning. These advantages make it preferable to use machine learning techniques to solve complex performance problems that are difficult for heuristics to find better solutions.

Chapter 4

Approximate Nearest Neighbor Search Performance Analysis

In this chapter, we present performance analysis of state-of-the-art approximate nearest neighbor search approaches, as a motivation for the proposed learned adaptive early termination. Finding the top-k nearest neighbors among database vectors for a query is a key building block to solve problems such as large-scale image search and information retrieval [67, 75, 91], recommendation (the candidate retrieval component) [27], entity resolution [52], and sequence matching [13]. As database size and vector dimensionality increase, exact nearest neighbor search becomes expensive and impractical due to latency and memory constraints [14, 15, 104]. To reduce the search cost, approximate nearest neighbor (ANN) search is used, which provides a better tradeoff among accuracy, latency, and memory overhead.

ANN search traditionally uses a mixture of *compression* and *indexing*. *Code compression* shrinks database vectors into compact codes, either binary [23, 44] or based on various quantization methods [7, 24, 57, 110]. These techniques can reduce computation latency and memory requirements by nearly an order of magnitude. With code compression alone, however, the search process remains exhaustive. The second form of approximation, the *indexing structure*, restricts the distance evaluation to a subset of elements. State-of-the-art approaches include inverted file index [8, 12, 57] which groups database vectors by clusters, and graph-based approaches [3, 31, 37, 79, 80] which perform beam search on proximity graphs.

The number of the database vectors to search (i.e., the search termination condition) affects performance: the more vectors to search, the higher the accuracy (good) and latency (bad). The optimal termination condition (minimum search to find the nearest neighbor) for each query is not obvious. As a result, state-of-the-art indexing approaches use various fixed configurations to apply the same search termination condition for all queries. For example, inverted file index could terminate after searching the top 5 nearest clusters for each query, and graph-based index could terminate after searching 100 neighboring graph nodes for each query.

In our study of three datasets, we find that the number of vectors that must be searched to find the ground-truth nearest neighbor varies widely among queries: it is possible to find the nearest neighbor for most queries by searching a small fraction of the dataset, but the remaining "difficult" queries require much more searching. For inverted file index, it is possible to reach 80% accuracy by only searching up to the top 1.20% nearest clusters, but reaching 95-100% accuracy requires searching up to the top 16.90% nearest clusters. For graph-based approaches, 80% accuracy can be obtained by searching up to 0.13% of total graph nodes, but reaching 95-100% accuracy requires searching up to 11.83% of total graph nodes. As a result, fixed configurations force 80% of queries to search an unnecessarily large number of database vectors, just to cover the remaining 20% "difficult" queries.

Based on the study we argue that it is necessary to apply different termination conditions for each query. One challenge is that static features such as the query vector itself are not sufficient to predict this termination condition. During our feature exploration, we find that runtime features such as intermediate search results after a certain amount of search (e.g. when reaching 60-80% accuracy) are effective in predicting how much more work should be performed for each individual query. These features enable us to build prediction models in the next chapter that achieve the same accuracy with less total amount of search compared to the fixed configurations.

For the rest of this chapter, Section 4.1 presents the background of approximate nearest neighbor search. Section 4.2 presents the performance analysis of state-of-the-art ANN search techniques.

4.1 Background

In this section, we first describe the ANN search problem and then introduce the state-of-the-art ANN indexing approaches.

ANN search problem. Nearest neighbor search is the problem of finding the vectors in a given set that are closest to a given query vector. As database sizes reach millions or billions of entries, and the vector dimension grows into the hundreds [9, 58], approximate nearest neighbor search becomes necessary in order to achieve a better tradeoff between accuracy and efficiency. Formally, the ANN search problem [94] is defined as:

Definition 4.1.1 Approximate Nearest Neighbor (ANN) Search Problem. *Let* $X = \{x_1, ..., x_N\} \in \mathbb{R}^D$ *represents a set of* N *vectors in a* D*-dimensional space and* $q \in \mathbb{R}^D$ *represents the* *query. Given a value $K \leq N$, ANN search finds the K closest vectors in X to q, according to a pair-wise distance function $d\langle q, x\rangle$, as defined below:*

$$TopK_q = \underset{x \in X}{\text{K-argmin}}\, d\langle q, x\rangle \tag{4.1}$$

The result is a set $TopK_q \subseteq X$ such that (1) $|TopK_q| = K$ and (2) $\forall\, x_q \in TopK_q$ and $x_p \in X - TopK_q : d(q, x_q) \leq d(q, x_p)$.

In this paper we use the Euclidean distance as the distance function to measure the (dis)similarity between vectors.

4.1.1 Compressed representation

The first source of approximation comes from compressed representations, originally proposed to improve search efficiency [104]. Later work proposed compact binary codes to improve image similarity search [23, 75]. Recent work uses "vector quantization", in which a vector is first reduced by principal component analysis (PCA) dimension reduction and then is subsequently quantized [42, 45, 57]. Although code compression introduces distance approximation error, it provides more efficient vector storage and distance calculation. However, the search remains exhaustive: all database vectors must be evaluated.

4.1.2 Specialized ANN indices

In this paper we focus on improving the efficiency of another approximation method: indexing. The ANN search index structure restricts the distance evaluations to a subset of database vectors. In this paper we focus on two state-of-the-art methods: inverted file index (IVF [57] and IMI [8]) and graph-based approaches (HNSW [79, 80]). There exist other indexing approaches including deterministic space partitioning such as kd-trees [22], and randomized indexing approaches based on locality sensitive hashing (LSH) [28, 41, 53, 54, 68, 113]. We believe that the idea of adaptive termination conditions also applies to those approaches.

Inverted file index

The inverted file (IVF) index is a variant of inverted index. Inverted indices were proposed in the computer vision community [97] and have long been used in the information retrieval community [81]. In a recent paper about product quantization (a vector compression technique) [57], the inverted file index is introduced as a nearest neighbor search index to avoid exhaustive search. The IVF index groups database vectors into different

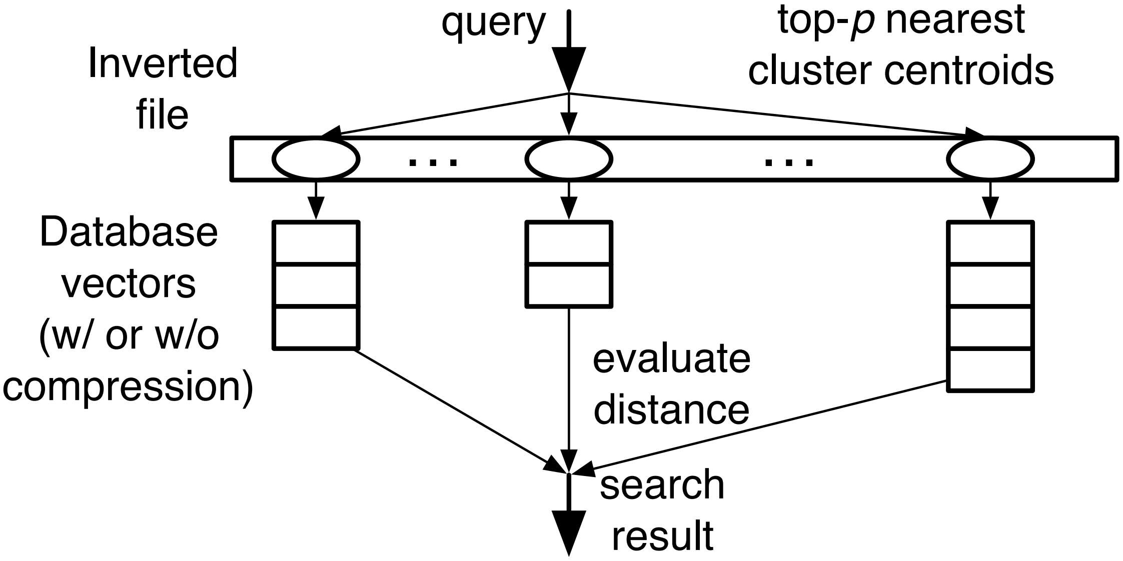


Figure 4.1: The IVF index.

clusters. When building the index, a list of cluster centroids is trained by K-means clustering, and each database vector is assigned to the cluster with the closest centroid. During searching, the index first computes the distances between the query and all cluster centroids, then evaluates the database vectors belonging to the top-p nearest clusters as shown in Figure 4.1. Using a larger p increases both accuracy (good) and search latency (bad). Different compression methods often use IVF to avoid exhaustive search. In those cases the database vectors are compressed into shorter codes.

Several follow-up projects aim to improve the inverted file indexing approach. Inverted multi-index (IMI) [8] decomposes the vectors into several subspaces and trains separate list of centroids in each subspace, which leads to a fine-grained space partition. Baranchuk *et al*. propose to build sub-clusters in each cluster to further restrict the number of database vectors to be evaluated [12]. However, all of IVF variants search the same fixed number of nearest clusters for all queries.

Graph-based approaches

One of the state-of-the-art graph-based indexing approaches is the hierarchical navigable small world graphs (HNSW) [79, 80]. This index includes multiple layers of proximity graphs where each graph node represents a database vector as plotted in Figure 4.2. The top layer contains only a single node and the base layer includes all database vectors. Each intermediate layer includes a subset of database vectors covered by the next lower layer.

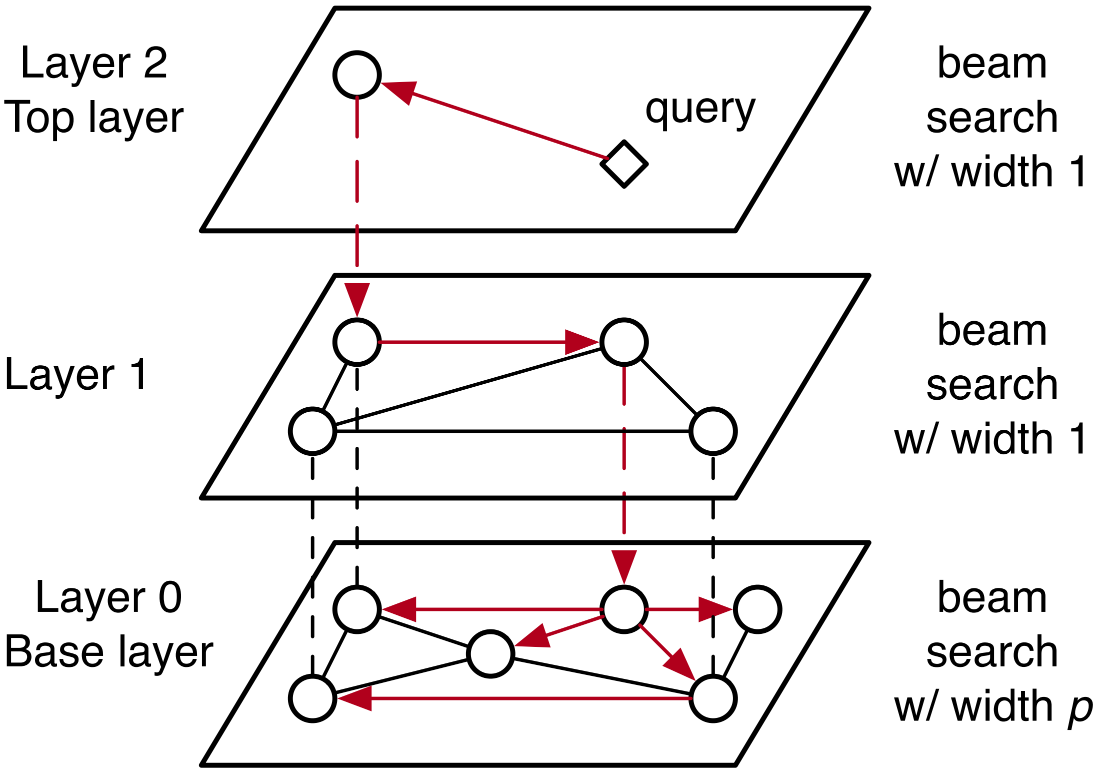


Figure 4.2: A three-layer HNSW index.

When building the index, database vectors are inserted one by one. Each vector is inserted into multiple layers from the base layer to a certain layer determined by an exponentially decaying probability distribution. At each insertion, the newly-inserted vector is connected to at most a fixed number of nearest nodes previously inserted to the same graph. As a result the graphs created in HNSW are *approximate* knn-graphs, since the connected neighbors might not be the ground truth nearest neighbors. In addition, HNSW's algorithm employs heuristics that connect some far away nodes from different isolated clusters to improve the global graph connectivity.

When handling a query, a variant of beam search with beam width 1 (higher layers) or p (base layer) is performed at each layer. Search starts from the top layer. At each layer (except the base layer), the neighboring nodes of the start node are evaluated, and the node nearest to the query is selected as the start node of the next layer. These 1-hop beam searches aim to converge to a base layer start node that is fairly close to the ground truth nearest neighbor of the query. At the bottom base layer, first the start node is inserted to an empty candidate priority queue where the priority is based on the distance to the query. Then at every iteration the algorithm pops the top candidate node from the queue, evaluates the distances between the query and popped node's neighbors, updates the current found best neighbor if necessary, and inserts popped node's neighbors to the candidate queue. Eventually top-p best candidate nodes (based on the distance to query) have their neighboring nodes evaluated. Like in IVF, a larger p increases both accuracy and latency.

Several follow-up projects aim to improve the proximity graph-based approach. Douze *et al.* propose to combine HNSW with quantization [31]. Navigating Spreading-out Graph (NSG) aims to reduce the graph edge density while keeping the search accuracy [36, 37]. SPTAG combines the IVF index and proximity graph for distributed ANN search [3, 101, 102, 103]. GRIP is a capacity-optimized multi-store ANN algorithm which combines the HNSW and IVF index to jointly optimize search time, memory usage, and accuracy with both DRAM and SSDs [108]. As with the IVF variants, all of these proximity graph variants employ fixed configurations to perform a fixed amount of graph traversal for all queries.

4.2 Performance analysis of state-of-the-art ANN search approaches

4.2.1 Fixed configurations lead to inefficient latency-accuracy tradeoff

To motivate our work, we first evaluate the baseline performance of existing indexing approaches under different fixed configurations. We explore three million to billion-scale datasets summarized in Table 4.1. DEEP is a dataset of CNN image representations with 1 billion base vectors and 10000 queries [9]. Each vector has 96 dimensions where each coordinate is a floating-point number between -1 and 1. SIFT is a dataset of local SIFT image descriptors with 1 billion base vectors and 10000 queries [58]. Each vector has 128 dimensions where each coordinate is an integer between 0 and 128. GIST is a dataset of global color GIST descriptor with 1 million base vectors and 1000 queries [57]. Each vector has 960 dimensions where each coordinate is a floating-point number between 0 and 1. Each dataset also includes a separate set of training vectors and we use them to train the prediction models for the proposed adaptive early termination technique.

In this experiment, for DEEP and SIFT we use the first 10M base vectors as the database. For GIST, we use all 1M base vectors as the database. We evaluate the CPU-only IVF and HNSW implementation in the Faiss similarity search library [60]. We build IVF indices without vector compression in this experiment. Following the standard approach, the number of clusters are configured close to the square root of the database size. For each query, database vectors belonging to the top-p nearest clusters will be evaluated, and we evaluate the performance under different p (in Faiss this parameter p is called *nprobe*

Dataset	Vector Dimension	Num. Base Vectors	Num. Training Vectors	Num. Query Vectors
DEEP	96	10M, 1B	1M	10000
SIFT	128	10M, 1B	1M	10000
GIST	960	1M	0.5M	1000

Table 4.1: Summary of explored datasets.

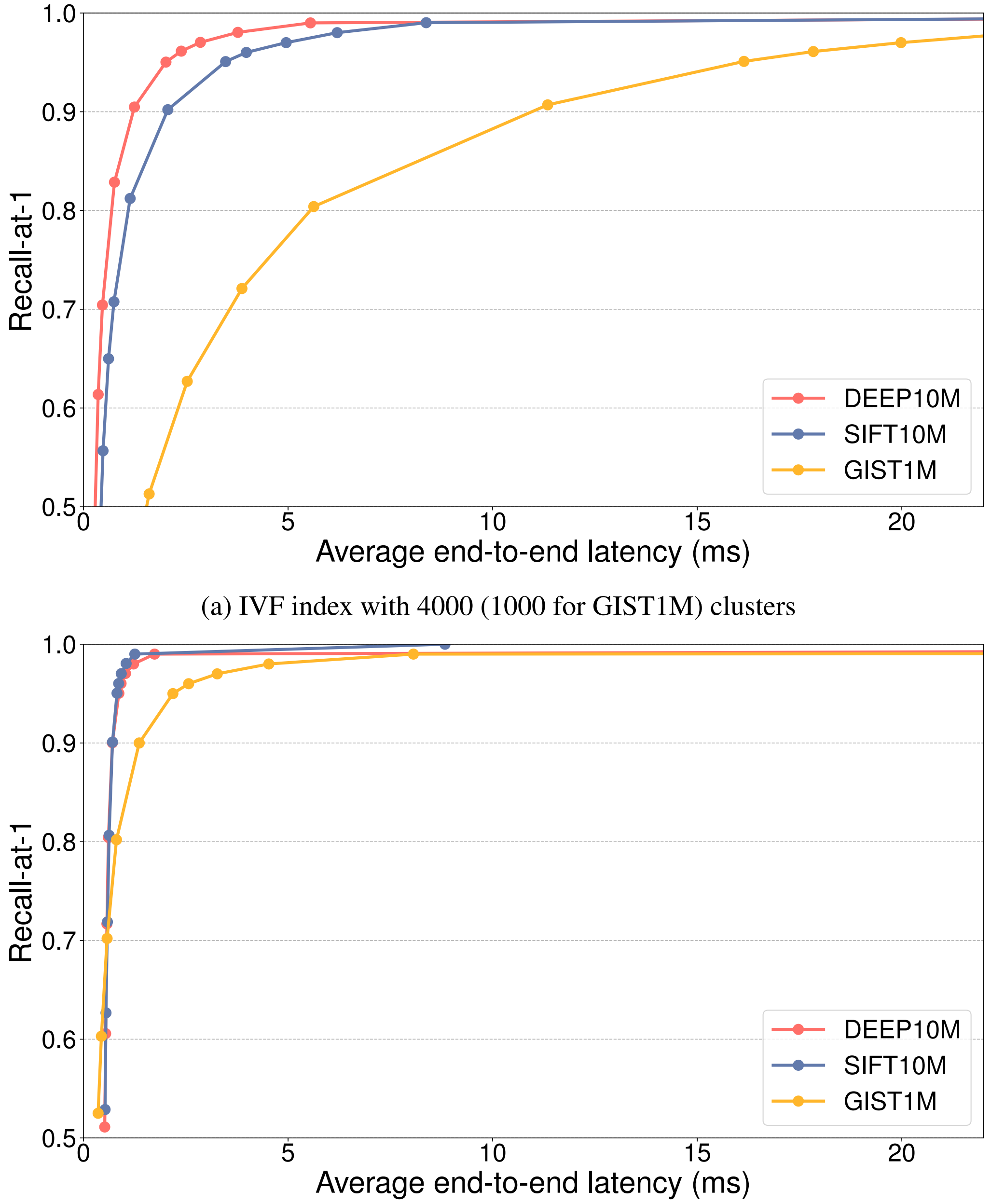
for IVF).

The HNSW index in Faiss uses parameters M and *efConstruction* to adjust the index complexity. For each database vector, the probability of inserting it into layer i graph is $(1/M)^i$, while the top layer always has only one node as the search starting point. Each inserted vector will have at most $2M$ connected neighbors in the base layer and at most M neighbors in other layers and the connections are determined by a beam search with width = *efConstruction*. We build HNSW indices with M = 16 and *efConstruction* = 500 based on the original HNSW work [80]. For each query, top-p best candidate nodes in the base layer have their neighboring nodes evaluated, and we evaluate the performance under different p (in Faiss this parameter p is called *efSearch* for HNSW).

We search for the top-1 nearest neighbor for each query and the accuracy is represented as recall-at-1 (the fraction of queries where the top-1 nearest neighbor returned from search is (one of) the ground truth nearest neighbor). One thing to note is that a query may have multiple ground truth nearest neighbors: one reason is that we find that all three datasets we use have duplicate base vectors. In that case we count the search successful as long as one of the ground truth is returned. Then we measure the recall and the average latency when using different fixed configurations (*nprobe* for IVF and *efSearch* for HNSW). The detailed methodology is described in Section 5.2.1.

Figure 4.3 illustrates the baseline performance of IVF and HNSW indices where each dot represents a different fixed configuration: For DEEP10M, it takes only 0.757 ms/0.610 ms on average to reach 0.8 recall on IVF/HNSW index, but it takes 2.015 ms/0.865 ms on average to reach 0.95 recall; For SIFT10M, it takes only 1.141 ms/0.625 ms on average to reach 0.8 recall on IVF/HNSW index, but it takes 3.474 ms/0.819 ms on average to reach 0.95 recall; For GIST1M, it takes only 5.628 ms/0.807 ms on average to reach 0.8 recall on IVF/HNSW index, but it takes 16.142 ms/2.185 ms on average to reach 0.95 recall. To reach recall targets above 0.95, some extreme cases take hundreds of milliseconds on average. This shows that the fixed configuration approach leads to undesirably high average latency when trying to reach high recall target.

Both IVF and HNSW have worse performance on GIST1M for two reasons: First,



(a) IVF index with 4000 (1000 for GIST1M) clusters

(b) HNSW index with M=16 and *efConstruction*=500

Figure 4.3: Baseline performance with fixed configurations: Average end-to-end latency at different recall-at-1 targets. Each dot represents a different fixed termination condition applied to all queries. Note the y-axis starts at 0.5.

Euclidean distance computation time is proportional to the vector dimension. Second, searching the GIST1M dataset is harder than SIFT10M and DEEP10M, despite its smaller

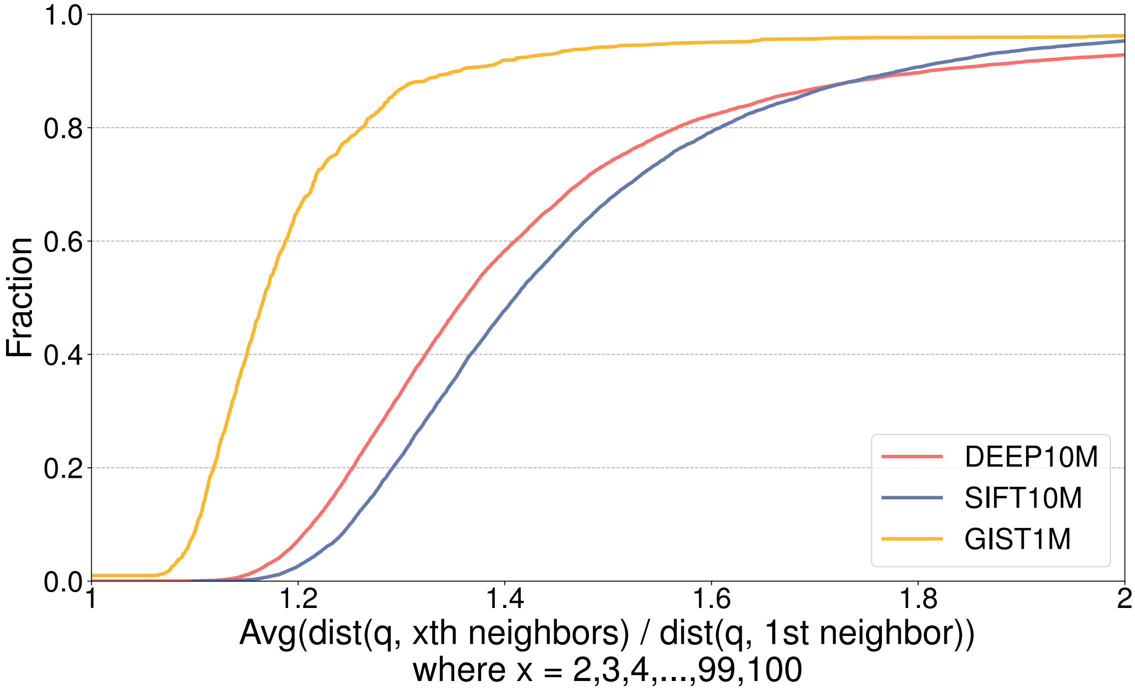


Figure 4.4: CDF of the average of ratios between dist(q, xth neighbors) and dist(q, 1st neighbor). Closer to 1 indicates that it is harder to find the 1st neighbor.

size. Figure 4.4 plots the CDF of average ratio between distance(query, 2nd to 100th neighbors) and distance(query, 1st neighbor) under exhaustive nearest neighbor search. When the average ratio is closer to 1, it means that the top 100 neighbors for each query are more similar to each other, which increases ANN search difficulty: For the IVF index, queries might be close to many more clusters; For HNSW index, it could take much more graph node traversal to reach the ground truth nearest neighbor.

4.2.2 Queries need different termination conditions

To investigate the reason for undesirably high average latency with fixed configurations, we must identify the minimum amount of search needed to find the ground truth nearest neighbor for each query. For the IVF index, the minimum amount of search is represented by the minimum number of nearest clusters to search (*nprobe*). For HNSW index, we do not use the number of searched top candidate nodes (*efSearch*), instead using the minimum number of distance evaluations to represent the minimum amount of search. This is because: 1) The distance evaluation between query and database vector is the time consuming task; 2) The number of distance evaluations varies greatly even with the same number of searched top candidate nodes: In some cases the searched candidate nodes are neighbors to each other, where many redundant distance evaluations are avoided. In some cases it is more like a depth-first search, where the number of evaluations is close to "number of searched top candidate nodes × number of connected neighbors per node". For a few queries in DEEP and GIST datasets, we are not able to find this minimum amount of search in HNSW index because the ground truth nearest neighbor is not found after searching all reachable graph nodes due to graph connectivity issue.

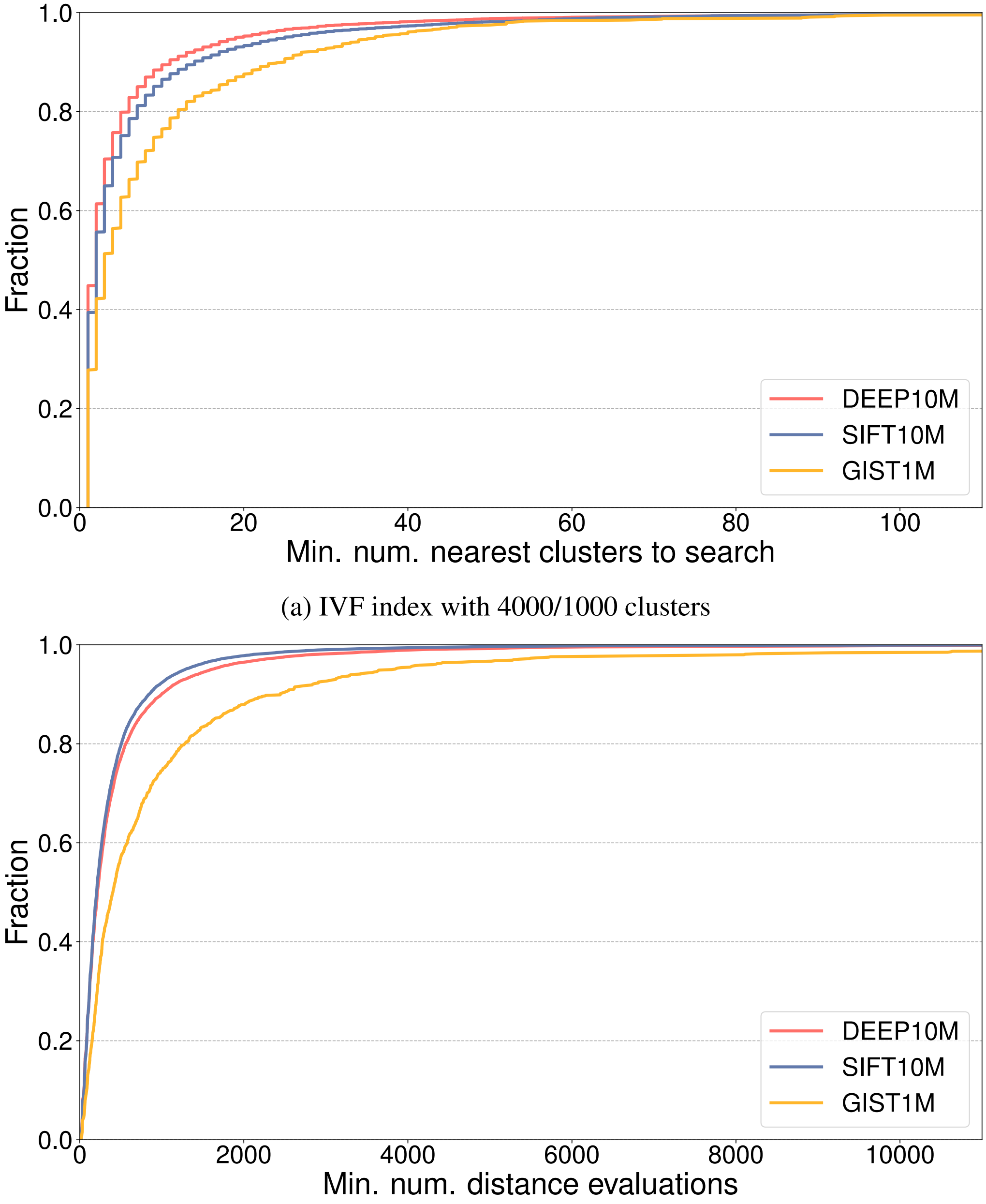
Figure 4.5 illustrates the CDF of minimum amount of search to find the ground truth nearest neighbor for each query: For IVF, 80% of queries only need to search at most top-6/7/12 nearest clusters for DEEP/SIFT/GIST, but the other 20% queries must search up to top-606/367/169 nearest clusters; For HNSW, 80% of queries only need to perform at most 547/481/1260 distance evaluations for DEEP/SIFT/GIST, but the other 20% queries require up to 88696/16618/118277 distance evaluations. We find the same trend when using the training vectors as query vectors, which shows that the training and query vectors share the same distribution.

4.2.3 How to predict the termination condition

To predict the termination condition for each query, we must identify relevant measurable features. Static features such as the query vector are helpful, but our study shows that it does not suffice: features obtained at the start of search do not accurately indicate the termination condition.

Instead, we find that the intermediate search result after a certain amount of search (e.g., when 60-80% of queries/training vectors have found their ground truth nearest neighbors) is a critical runtime feature that indicates how much more work should be performed for each query. For the IVF index, we measure the 50th, 75th, and 90th-percentile minimum number of nearest clusters to search for different ranges of distance between query and intermediate 1st neighbor after searching top 6 (DEEP)/7 (SIFT)/12 (GIST) nearest clusters. For the HNSW index, we measure the 50th, 75th, and 90th-percentile minimum number of distance evaluations for different ranges of distance between query and intermediate 1st neighbor after 547 (DEEP)/481 (SIFT)/1260 (GIST) distance evaluations.

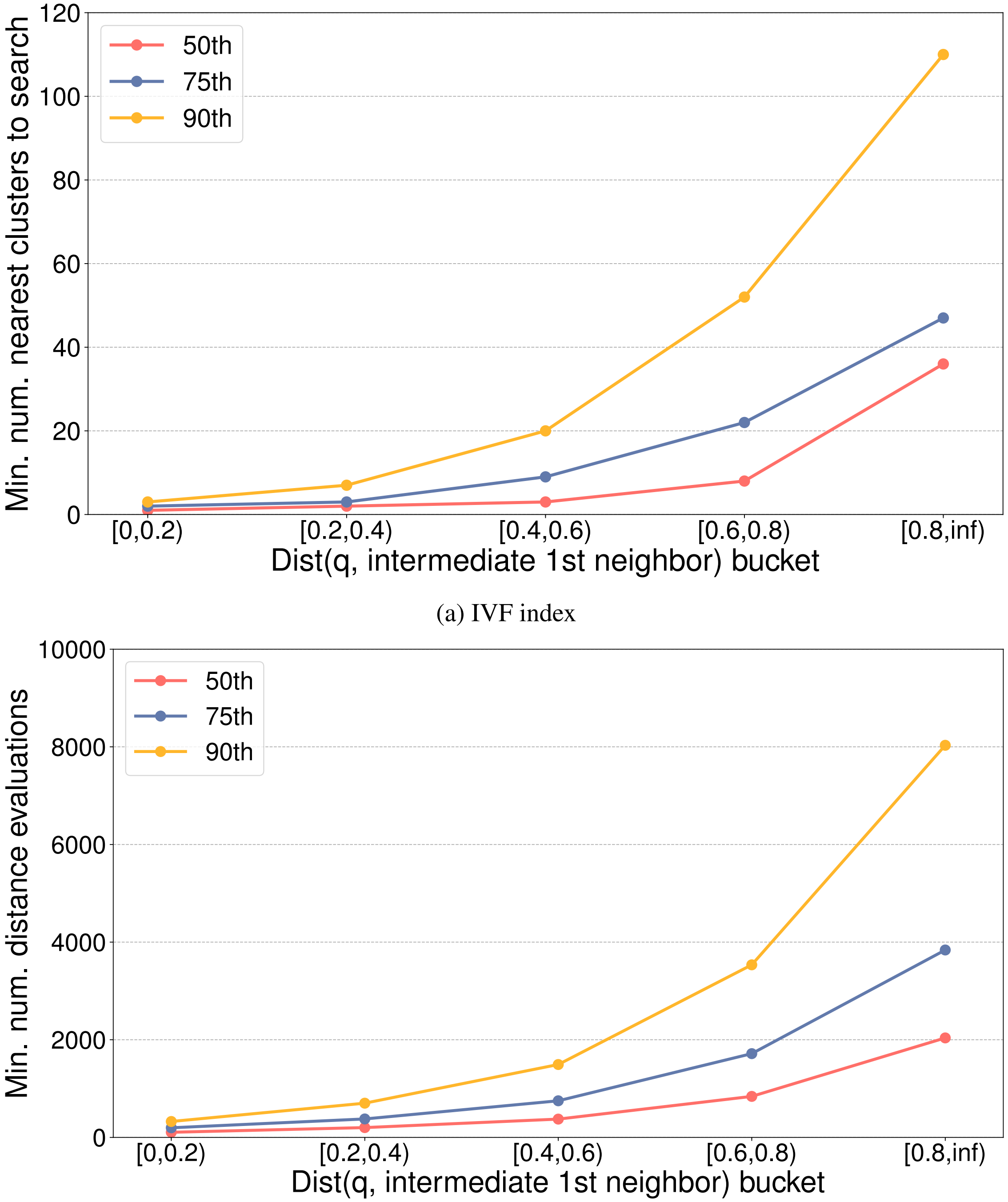
Figure 4.6 illustrates this relationship for the DEEP dataset (similar trends are found in SIFT and GIST). As the distance between query and intermediate search result increases, the minimum amount of search to find the ground truth also increases. This shows that intermediate search results are highly relevant features: if your search result is still far away from the query, you probably want to search more. To get this feature, we need to search a fixed amount for all queries, even though some of them need less than that. However we argue that this runtime feature is necessary for the prediction model as explained in Section 5.1, and the majority of the variation among search termination conditions is still remained to be exploited by the proposed approach.



(a) IVF index with 4000/1000 clusters

(b) HNSW index with M=16 and *efConstruction*=500

Figure 4.5: CDF of minimum amount of search to find the ground truth nearest neighbor for each query.



(a) IVF index

(b) HNSW index

Figure 4.6: DEEP10M: 50th/75th/90th-percentile minimum amount of search for different ranges of distance between query and intermediate 1st neighbor.

Chapter 5

Learned Adaptive Early Termination for Approximate Nearest Neighbor Search

In the last chapter, we show that state-of-the-art ANN approaches use fixed configurations that apply the same termination condition (the size of subset to search) for all queries, which leads to undesirably high latency when trying to achieve the last few percents of accuracy.

To achieve a better tradeoff between latency and accuracy, we propose a novel approach that adaptively determines search termination conditions for individual queries. To do so, we build and train gradient boosting decision tree models [35] (using the LightGBM framework [62]) and neural networks models [95] (using the PyTorch framework [89]) to learn and predict when to stop searching for each query for three indexing approaches: IVF [57], HNSW [80], and IMI [8]. We implement our approach over the Faiss similarity search library [60], and evaluate the end-to-end performance on three million to billion-scale datasets (DEEP10M & DEEP1B [9], SIFT10M & SIFT1B [58], and GIST1M [57]).

Without vector compression and for applications targeting 95 to 100% recall-at-1 accuracy, our approach consistently reduces end-to-end latency vs. using fixed configurations on three million-scale datasets (DEEP10M, SIFT10M, GIST1M): For the IVF index, the average latency is reduced by up to 63% (2.7 times speedup); For the HNSW index, the average latency is reduced by up to 89% (9.4 times speedup).

With OPQ vector compression [42] and for applications targeting 95 to 100% recall-at-100 accuracy, our approach consistently reduces end-to-end latency vs. using fixed configurations. For the IVF index+OPQ (DEEP10M, SIFT10M, GIST1M), the average latency is reduced by up to 52% (2.1 times speedup). For the IMI index+OPQ (DEEP1B, SIFT1B), the average latency is reduced by up to 59% (2.4 times speedup).

For the rest of this chapter, Section 5.1 describes the design of the proposed prediction models. Section 5.2 describes the experimental methodology and reports the results. Section 5.3 presents the related work. Section 5.4 concludes the learned early termination work.

5.1 Design

In this section, we lay out the way that our approach is trained and integrated into both IVF and HSNW. Our predictor takes a set of inputs from the algorithm reflecting the current query state, and outputs a numerical value indicating how much more work should be done. We begin by describing the parameters that our predictor accepts and how it is trained. We then discuss the integration into the indices themselves. All the result numbers in this section are for the DEEP10M, SIFT10M, GIST1M datasets with IVF and HNSW indices without vector compression. We follow the same training methodology for the cases with billion-scale datasets and/or OPQ vector compression.

5.1.1 General workflow

The output

For each query, we want to predict the minimum amount of work to reach the ground truth nearest neighbor (i.e., a regression model). Different indexing approaches may have different metrics, but what we need is a numerical value that is proportional to the search latency.

The inputs

Our study shows that the following three categories of features improve the prediction accuracy:

Query vector Since there could exist intrinsic relevance between minimum amount of search and query distribution, we use the query vector as the first kind of features where each dimension is a single feature.

Index structure Different indices have different metrics to describe how far the query is to a certain sub-region of database. This can help us to understand whether it is likely that the nearest neighbor belongs to a certain region.

Intermediate search results As motivated in Section 4.2.3, we find that the intermediate search result as a runtime feature demonstrates high relevance to what we want to predict.

Training and tuning

We use the training vectors in Table 4.1 to generate training/validation data and use the query vectors to generate testing data. Each vector generates one row of data which includes both the output target value and the input features. For the output value, we need to first perform an exhaustive search to find the ground truth nearest neighbor(s), then find the minimum amount of search to reach (one of) it. Based on the output metric, different indices have different ways to find this minimum amount. For the input features, we can compute the index structure feature based on the training/query vector and the index. We can compute the intermediate search results feature by performing the desired fixed amount of search.

Finding the ground truth via exhaustive search may take up to 13 hours on a single Nvidia GeForce GTX 980 GPU with 1 billion database vectors and 1 million training vectors. This exhaustive search is a one-time cost amortized across all online/offline queries as long as there is no change to the database and training vectors. From experience at Microsoft Bing, there could be hundreds of millions of latency-sensitive online web search queries per day (that require ANN search) [70, 72]. Thus the exhaustive search is a small cost compared to the total latency and computation reduction that the proposed approach can achieve over all queries. If there is any change to the database/training vectors, we can incrementally update the ground truth which takes much less time than the initial computation. Last but not least, it is possible to greatly reduce this search time by distributing the search to multiple GPUs/machines since it is a parallelizable offline computation.

Model 1: Gradient boosting decision trees First we elect to use gradient boosting decision trees for the prediction models. We build and train them using the LightGBM library [62]. Gradient boosting decision trees [35] are an ensemble model of decision trees. Similar to the reasons described in the caching work, we select this model because of several of its strengths: Both training and inference are fast, and the models allow for introspection. Training takes only 5 to 39 seconds on a CPU with 1 million training entries and 100 iterations. Inference takes only tens of microseconds and the model is only hundreds of KB in size (although these are proportional to the number of training iterations), which is a small latency/memory overhead. Importantly, decision tree models allow us to identify the importance of individual features by the total error reduction per split in the tree, which is helpful during feature exploration and for explaining why the system works.

LightGBM's documentation includes instruction on parameters tuning [4]. We use the default decision tree structure parameters. As described above, the number of training iterations affects the model size and the prediction accuracy/latency. To balance the tradeoff, we choose a relatively small number of training iterations (100) and high learning rate (0.2) (except the case of billion-scale dataset where we use a larger training iterations (500) and smaller learning rate (0.05)). Since it is important to identify those queries that need much more search, we choose to minimize the L2 loss function, which favors the outliers.

Model 2: Neural networks We also explore the use of neural networks-based prediction models for search termination prediction. Neural networks is generally applied as a powerful machine learning technique for pattern recognition and has attained outstanding accuracy in many computer vision and natural language processing tasks [20, 66, 83]. However, there have been very few studies of adopting neural networks-based prediction for ANN index building in high dimensional vector space. One of the main reasons is that neural networks has been traditionally thought to be quite computationally expensive. However, recent advances of hardware accelerators have demonstrated remarkable performance for neural networks computation [25, 48], and studies have also shown that even commodity hardware can provide significant latency boost for neural networks prediction with a careful design taking consideration of cache locality and parallelism [109].

We choose simple feed forward neural networks as our prediction model for its simplicity and flexibility. We use the same input features, output values, and training/testing data as the decision trees case. The neural networks model has a tunable number of hidden layers, where each has a tunable number of hidden neurons associated with it. For this study, we choose two hidden layers with 64 hidden neurons each, which we find to provide the best balance between prediction accuracy and overhead. We choose ReLU as the nonlinear activation function [85, 107].

We implement the model in PyTorch framework [89] and train the model on a Nvidia GeForce GTX 980 GPU. We use stochastic gradient descent (SGD) [17] as our optimization method to minimize the mean squared error loss function:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} |p_n - g_n|^2 \tag{5.1}$$

where θ denotes the neural networks parameters to be learned, p_n represents the predicted termination condition, and g_n represents the ground truth termination condition. We train the model over the training vectors with a mini-batch size of 32 for 10 epochs, with batch normalization at each layer [55]. We set the learning rate to be 0.00001 [17] and apply dropout to hidden layers with a dropout ratio of 0.1 [98]. The model weights are initialized with He initialization [49], and we set the weight decaying rate to 0.01 [51].

For both the decision trees and neural networks models, we perform the prediction/inference on CPU during evaluations. This is because this work target CPU-based ANN search, which has lower cost than current GPU-based ANN search thus preferable in online deployment. However, since machine learning models have very different performance characteristics on GPUs and customized hardwares [61], it would be an interesting direction to consider the learned search termination on different hardwares.

Integration and online prediction

To integrate the prediction model into the Faiss baseline, we first load the prediction model. Then for each query we perform a fixed amount of search until the intermediate search results are ready. Then we gather all the features and perform the prediction which produces the termination condition. If the predicted termination condition has passed, we stop immediately. Otherwise we keep searching until the termination condition is met.

5.1.2 The IVF index case

The output

For the IVF index (and the IMI variant), we build a regression model to predict the minimum number of nearest clusters to search. This number is the *nprobe* parameter, now determined for each query.

The inputs

We investigate 6 kinds of features summarized in Table 5.1. We use the ratios of distances between query and various nearest cluster centroids as the index structure features. The intuition is that if the query has similar distance to many cluster centroids, we probably want to search more clusters. We use the other four features to represent the intermediate search results. First we use the distances between the query and the 1st&10th neighbor after searching the top 6 (DEEP)/7 (SIFT)/12 (GIST) nearest clusters as two features. How much should we search before using the results as features is a hyperparameter. We explain how to tune it by grid search in Section 5.1.2. Then we use the ratio between the two features as another feature. Finally, we use the ratio between distance to the intermediate 1st neighbor and distance to the 1st nearest cluster centroid as the last feature. These features all aim to represent how good the intermediate search results are.

Feature	Description
F0: query	The query vector Each dimension is a single feature
F1: c_xth_to_c_1st (10 features)	Dist(q, xth nearest cluster centroid) / Dist(q, 1st nearest cluster centroid) where x $\in \{10, 20, 30, ..., 90, 100\}$
F2: d_1st	Dist(q, 1st neighbor after a certain fixed amount of search)
F3: d_10th	Dist(q, 10th neighbor after a certain fixed amount of search)
F4: d_1st_to_d_10th	F2: d_1st / F3: d_10th
F5: d_1st_to_c_1st	F2: d_1st / Dist(q, 1st nearest cluster centroid)

Table 5.1: IVF index input features.

Training and tuning

To build the training/testing data, the output target value (minimum number of nearest clusters to search) is generated by computing the distances between query and all cluster centroids to find the rank of the cluster where the ground truth nearest neighbor belongs to. The input features are generated by performing the actual search until the intermediate search results features are ready.

To find which features are more important, we use the per-feature gain stats from gradient boosting decision tree models where the importance of a feature is proportional to the total error reduction contributed by the feature. Table 5.2 summarizes the normalized feature importance (note that we combine the importance of multiple features for the first two kinds of features). The query vector contributes to roughly one third of the overall importance. The ratios of distances between query and nearest cluster centroids are also relevant as expected: when the ratios are closer to 1, the prediction value is higher. The other four intermediate results features contribute to another great portion of importance.

Table 5.3 summarizes the testing data accuracy when training with all features or only the query vector (5-fold cross validation on the training data produces similar accuracy) using both decision trees and neural networks (DEEP10M dataset) models. For all metrics, lower is better. The MAPE numbers show that our model achieves similar accuracy among the 3 datasets. Accuracy drops when training with only the query. This shows that using the intermediate search results as runtime features is critical to the prediction accuracy. For DEEP10M dataset, the neural networks model provides higher accuracy than the decision trees model.

Importance	DEEP10M	SIFT10M	GIST1M
F0: query	44.08%	31.60%	37.92%
F1: c_xth_to_c_1st	13.60%	18.64%	25.88%
F2: d_1st	31.98%	31.16%	0.25%
F3: d_10th	0.50%	0.41%	0.12%
F4: d_1st_to_d_10th	5.78%	12.85%	31.80%
F5: d_1st_to_c_1st	4.06%	5.34%	4.03%

Table 5.2: IVF index feature importance.

	MAE	MAPE	RMSE
DEEP10M, all features, decision trees	4.74	149%	16.01
DEEP10M, query only, decision trees	5.42	209%	17.22
DEEP10M, all features, neural networks	4.36	127%	15.75
SIFT10M, all features, decision trees	5.15	162%	12.72
SIFT10M, query only, decision trees	5.98	217%	13.66
GIST1M, all features, decision trees	7.68	220%	14.43
GIST1M, query only, decision trees	8.87	296%	15.56

Table 5.3: IVF index: mean absolute error, mean absolute percentage error, and root mean squared error of the regression model with different feature sets.

To illustrate the prediction accuracy, Figure 5.1 plots the average number of searched clusters v.s. the recall-at-1 for DEEP10M. We find similar trends in SIFT10M and GIST1M. For baseline, each dot on the line represents a different fixed configuration. For our approach, the number of nearest clusters to search equals *max(max_thresh, multiplier*prediction)*, where the *max_thresh* equals the maximum target value in the training data. When using all features, we also take the *min* between the value above and 6 (DEEP)/7 (SIFT)/12 (GIST) (the amount of search needed for the intermediate search results). Each dot on the line represents a different *multiplier*. Instead of adding an absolute value to the prediction value, we find that it's more efficient to multiply a coefficient since the distribution of target values is highly skewed.

Results show that the adaptive prediction-based approach consistently reduces the average number of searched clusters to reach the same recall targets compared with the baseline. When training with only the query, the performance drops but is still better than the baseline. With higher prediction accuracy, the neural networks model is able to search the least number of clusters to reach the same accuracy, especially at the accuracy target of 1.00. One thing to note that this is not the end-to-end performance measurement since factors like the prediction overhead are excluded. We will evaluate the end-to-end

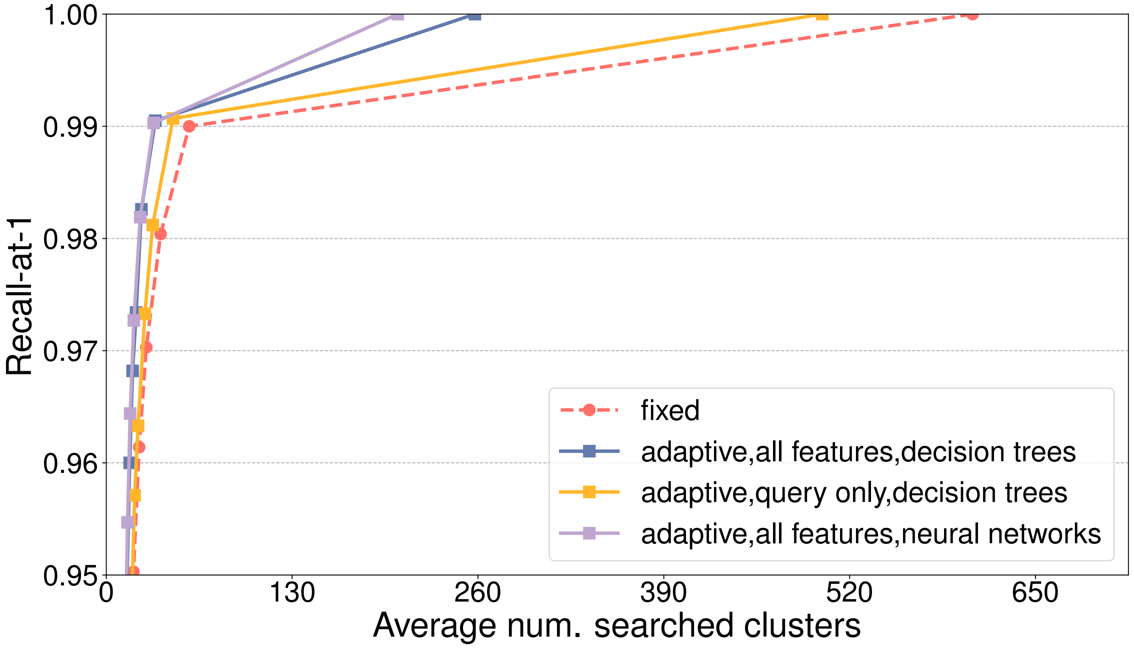


Figure 5.1: DEEP10M, IVF index: Average number of searched clusters vs. recall-at-1. Note the y-axis starts at 0.95.

performance in Section 5.2.

As mentioned previously how much should we search before using the intermediate results as features is a hyperparameter. Figure 5.2 illustrates how to perform grid search to tune this hyperparameter for DEEP10M for the decision trees model. If we search less before the feature generation, the intermediate result feature may provide less information gain, reducing the prediction accuracy. If we search more before the feature generation, all queries must search more, increasing the end-to-end average latency. This is why we choose the intermediate result after searching top-6 clusters which provides the best overall performance. To tune this hyperparameter for different datasets and/or different indexing approaches, we just need to perform a similar grid search on different intermediate search results that reach different recall accuracy targets.

Integration and online prediction

Algorithm 3 summarize how to integrate the prediction model into the IVF index. First we search a fixed number of top nearest clusters. Then we perform the prediction based on the query, the query-centroid distance ratios, and the intermediate search results. If the prediction value is larger than the fixed amount, we perform the remaining searching. At last we return the search results.

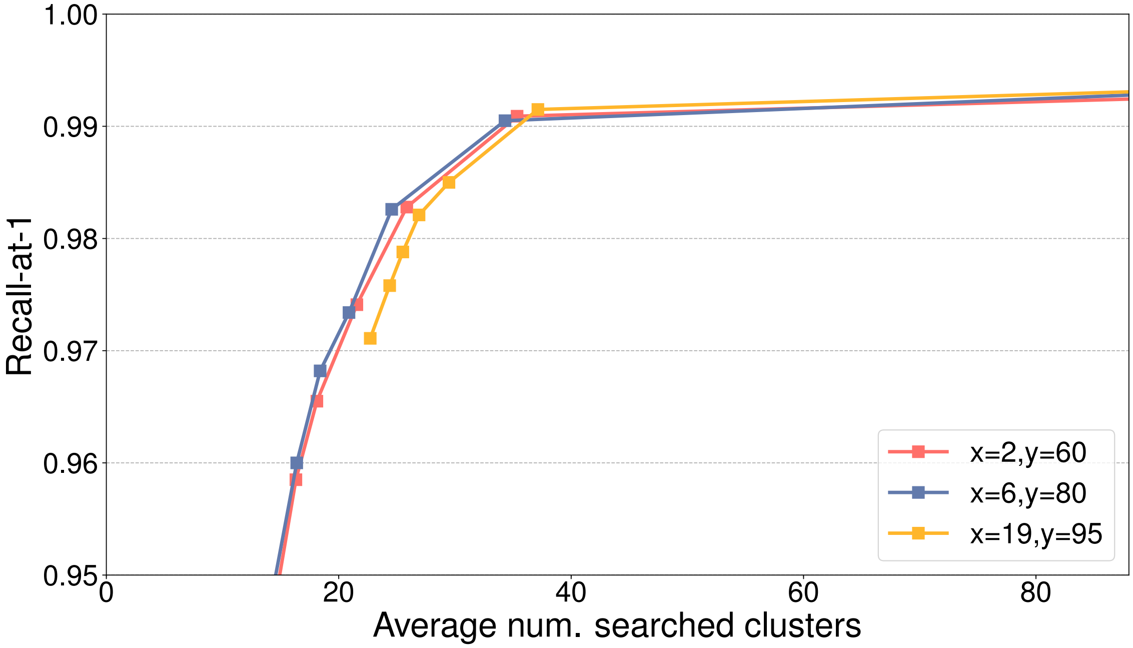


Figure 5.2: DEEP10M, IVF index: Grid search on finding the best intermediate search result features. Each line represents using the intermediate search results after searching the top-x nearest clusters and reaching y% recall accuracy on the training data.

Algorithm 3: Integration for the IVF index

```
input  : Query vector: $q$,
         number of neighbors to return: $k$,
         fixed amount to search before prediction: $f$.
output: List of top-$k$ nearest neighbors.
$h \leftarrow$ empty max heap with size $k$
sort clusters based on query-centroid distance
search the top $f$ nearest clusters and store the results in $h$
// beginning of the proposed approach
$input \leftarrow$ the input features including $q$, query-centroid distance ratios,
  intermediate search results from $h$
$p \leftarrow$ predict($input$)
if $p > f$ then
    search the top $(f+1)$th to $p$th nearest clusters and store the results in $h$
end
// end of the proposed approach
return top-$k$ nearest neighbors in $h$
```

Feature	Description
F0: query	The query vector Each dimension is a single feature
F1: d_start	Dist(q, base layer start node)
F2: d_1st	Dist(q, 1st neighbor after a certain fixed amount of search)
F3: d_10th	Dist(q, 10th neighbor after a certain fixed amount of search)
F4: 1st_to_start	F2: d_1st / F1: d_start
F5: 10th_to_start	F3: d_10th / F1: d_start

Table 5.4: HNSW index input features.

5.1.3 The HNSW index case

The output

For the HNSW index, we build a regression model to predict the minimum number of distance evaluations in the base layer. As explained in Section 4.2.2, this value is related but not equivalent to the *efSearch* parameter.

The inputs

We investigate 6 kinds of features summarized in Table 5.4. We use the distance between the query and base layer start node as the index structure feature, since it indicates the distance between the start node and the ground truth nearest neighbor. We use the other four features to represent the intermediate search results. We use the distances between the query and the 1st&10th neighbor after 368 (DEEP)/241 (SIFT)/1260 (GIST) distance evaluations as two features. Again how much should we search before using the results as features is a hyperparameter that can be tuned in the same fashion as the IVF case. Then we use the ratios between the two features and the distance-to-start-node feature as the last two features.

Training and tuning

To build the training/testing data, the output target value (minimum number of distance evaluations in the base layer) is generated by performing the actual ANN search until the ground truth nearest neighbor is found. Due to HNSW graph connectivity issues, we are not able to find the ground truth for a few vectors after evaluating all reachable nodes.

Importance	DEEP10M	SIFT10M	GIST1M
F0: query	13.39%	8.17%	27.65%
F1: d_start	1.02%	3.47%	1.26%
F2: d_1st	59.23%	69.07%	29.38%
F3: d_10th	4.81%	5.37%	0.74%
F4: 1st_to_start	6.11%	3.44%	18.80%
F5: 10th_to_start	15.43%	10.48%	22.17%

Table 5.5: HNSW index feature importance.

	MAE	MAPE	RMSE
DEEP10M, all features, decision trees	305	91%	1255
DEEP10M, query only, decision trees	348	119%	1311
DEEP10M, all features, neural networks	301	86%	1170
SIFT10M, all features, decision trees	231	83%	615
SIFT10M, query only, decision trees	268	111%	665
GIST1M, all features, decision trees	943	121%	4828
GIST1M, query only, decision trees	1011	155%	4877

Table 5.6: HNSW index: MAE, MAPE, and RMSE of the regression model with different feature sets.

So we exclude them from training data/regard as always missed for testing data. Since the range of number of distance evaluations for HNSW is much larger than the range of number of clusters to search for IVF as plotted in Figure 4.5, we use the base 2 logarithm of the number as the actual target value to help the training converge.

Table 5.5 summarizes the feature importance. The query vector again contributes to a fair portion of the overall importance. The distance between query and base layer start node contributes to a small amount of overall importance because it is dominated by the intermediate search results. The other four intermediate results features contribute to most of the overall importance.

Table 5.6 summarizes the testing data accuracy when training with all features or only the query (5-fold cross validation on the training data produces similar accuracy) using both decision trees and neural networks (DEEP10M dataset) models. The MAPE numbers show that our model achieves similar accuracy as the IVF case.

Figure 5.3 plots the average number of distance evaluations v.s. the recall-at-1 for DEEP10M. We find similar trends in SIFT10M and GIST1M. For baseline, each dot on the line represents a different fixed configuration. For our approach, the number of distance

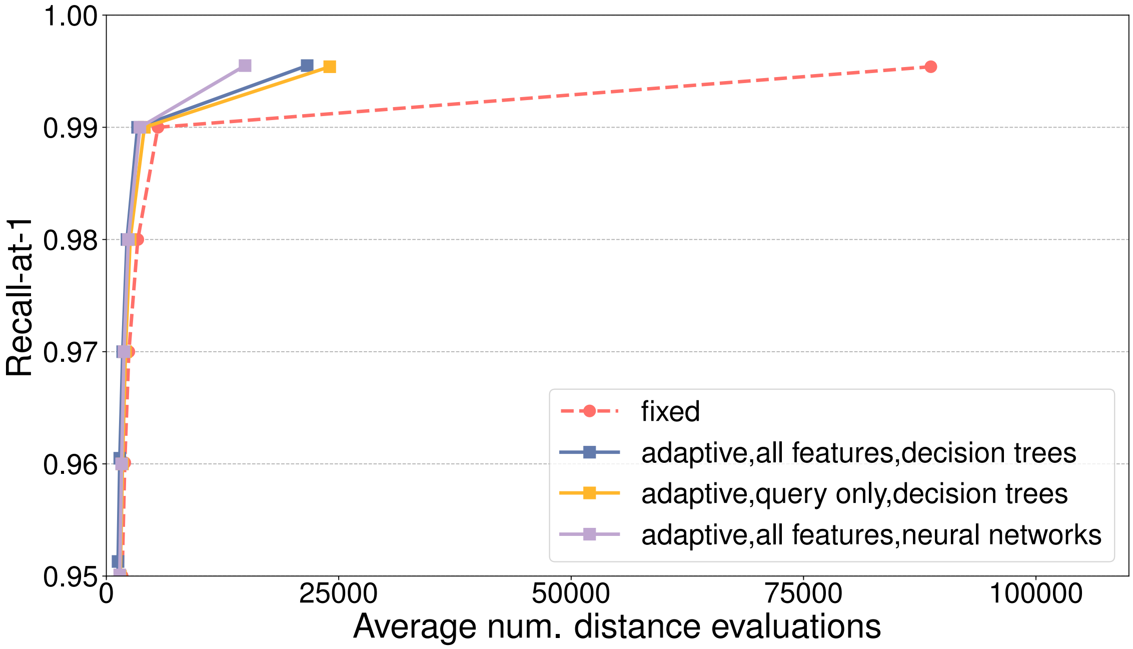


Figure 5.3: DEEP10M, HNSW index: Average number of distance evaluations vs. recall-at-1. Note the y-axis starts at 0.95.

evaluations equals *max(max_thresh, multiplier*2^prediction)*, where the *max_thresh* equals the maximum ground truth value in the training data. When using all features, we also take the *min* between the value above and 368 (DEEP)/241 (SIFT)/1260 (GIST) (the amount of search needed for the intermediate search results).

Results show that the adaptive prediction-based approach again consistently reduces the average number of distance evaluations to reach the same recall targets compared with the baseline. When training with only the query, the performance again drops but is still better than the baseline. With higher prediction accuracy, the neural networks model is able to search the least number of clusters to reach the same accuracy, especially at the highest accuracy target of 0.9955.

Integration and online prediction

Algorithm 4 summarize how to integrate the prediction model into the HNSW index. First we start from the top layer and perform beam search to reach the base layer. Then we perform beam search with unlimited beam width up to a fixed number of distance evaluations. Then we perform the prediction based on the query, the query-start node distance, and the intermediate search results. If the prediction value is larger than the fixed amount, we perform the remaining searching. At last we return the search results.

```
Algorithm 4: Integration for the HNSW index
input  : Query vector: $q$,
         number of neighbors to return: $k$,
         fixed amount to search before prediction: $f$,
         HNSW graphs: $G$,
         HNSW top layer start node: $s$.
output: List of top-$k$ nearest neighbors.
$h \leftarrow$ empty max heap with size $k$
while base layer in $G$ not reached do
    $s \leftarrow$ beam search with width 1 at the current layer starting from node $s$
    go to the next layer in $G$
end
$d \leftarrow$ distance between $q$ and $s$
$cnt \leftarrow 0$ // number of distance evaluations
while $cnt < f$ do
    perform beam search with unlimited width at base layer starting from node $s$
      and store the results in $h$
    increment $cnt$
end
// beginning of the proposed approach
$input \leftarrow$ the input features including $q$, query-start node distance $d$, intermediate
  search results from $h$
$p \leftarrow$ predict($input$)
if $p > f$ then
    while $cnt < p$ do
        perform beam search with unlimited width at base layer starting from node
          $s$ and store the results in $h$
        increment $cnt$
    end
end
// end of the proposed approach
return top-$k$ nearest neighbors in $h$
```

5.2 Evaluation

The evaluation section aims to compare the end-to-end performance achieved by three ANN indexing approaches (IVF, HNSW, IMI) with both fixed configurations and adaptive predictions. We first perform evaluation *without* compression because vector compression is orthogonal to the proposed adaptive early termination technique. We then measure the effectiveness of the proposed method *with* compression, because compression is often enabled to support large-scale ANN search. Section 5.2.1 describes the experimental methodology. Section 5.2.2 and 5.2.3 report the results of IVF and HNSW indices without compression (DEEP10M, SIFT10M, and GIST1M datasets). Section 5.2.4 and 5.2.5 report the results of IVF (DEEP10M, SIFT10M, and GIST1M datasets) and IMI (DEEP1B and SIFT1B datasets) indices with OPQ vector compression [42]. Section 5.2.6 discusses the effect of batching.

In this section we mainly evaluate the decision trees-based prediction model (one prediction per query) due to its faster training and tuning time. However, in Section 5.2.2 and 5.2.3 we also presents two additional approaches: one using multiple decision tree predictions per query instead of a single prediction; one using the neural networks-based prediction model to determine the termination condition.

5.2.1 Methodology

Setup. We implement our prediction-based approach (using the models trained with all features) in the Faiss similarity search library (CPU version) [60], and compare with the fixed configuration baseline approach as evaluated in Section 4.2.1. All experiments are executed on a machine with Intel® Xeon® E5-2680 v2 (2.8 GHz) processor and 128 GB of memory.

Prediction overhead. For the memory overhead of the prediction, we have one prediction model per indexing type and per dataset with sizes 247-310 KB for decision trees and 57-97 KB for neural networks, which is much smaller compared to the index and data size. When making a prediction, a temporary array is allocated to gather the features.

For decision trees model, each prediction takes between 7 us and 47 us depending on the indexing type and vector dimension. The number of input features and the number of training iterations affect the prediction overhead. When the dataset size increases, we may need to increase the number of training iterations in order to keep prediction accuracy high. For our experiments with 1 billion database vectors, we increase the number of training iterations from 100 to 500, which increases the prediction overhead by 5 times. But this is still beneficial since the overall search also takes longer. The number of neighbors to return (the k) does not affect the prediction overhead, since our model predicts the minimum termination condition to find the top-1 neighbor.

For neural networks model, each prediction (on CPU since we evaluate CPU-only ANN search) takes 278 to 322 us depending on the indexing type and vector dimension. This is slower than the decision trees model mainly because neural networks require heavier computations such as matrix multiplication. However, there exist many optimization techniques to reduce neural networks prediction overhead that we haven't explored, such as quantization and CPU vectorization. On the other hand, our evaluation shows that even with the current higher prediction overhead, the neural networks-based approach is able to provide the best performance on certain accuracy targets.

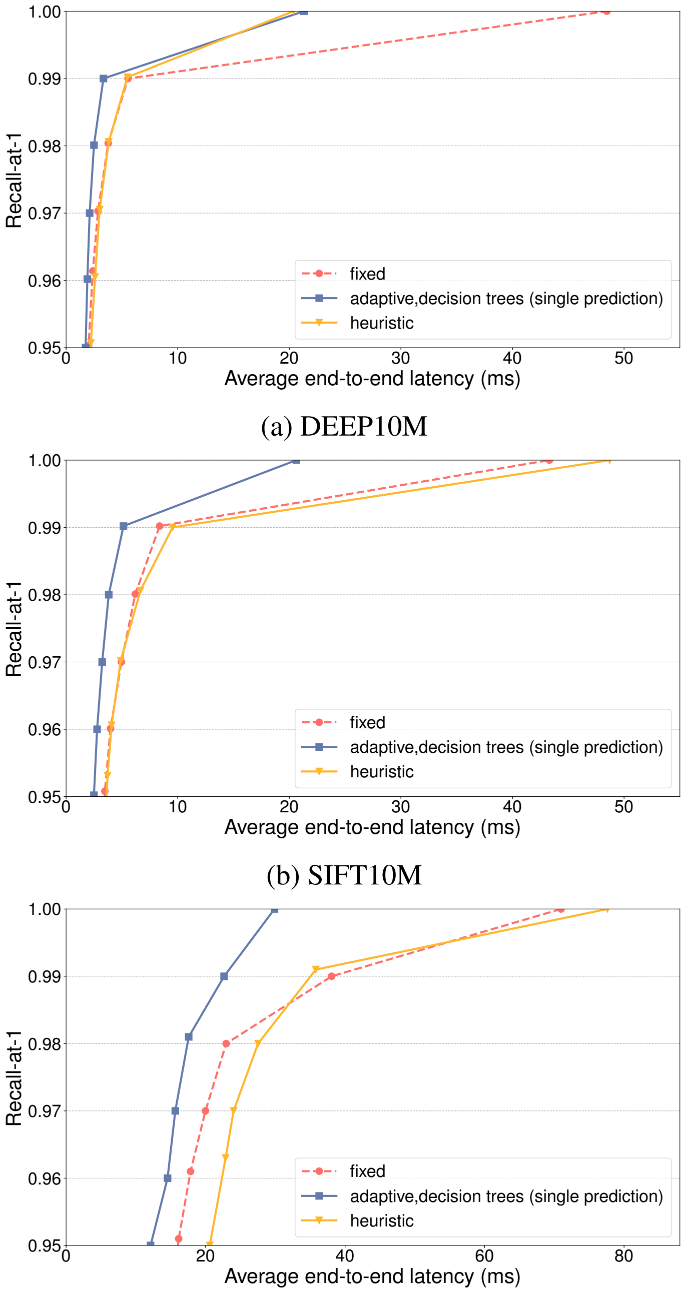
Performance metric. We envision that both our technique and the baseline approach would generally be deployed with a per-application expected accuracy target. This accuracy is *expected*, because no ANN search technique can guarantee a specific accuracy target without knowing the ground truth answer (in which case it could simply return the optimal value). The systems can, however, meet an expected accuracy target if the online query distribution matches the distribution of the training query vectors. This accuracy target can be met by appropriately configuring various parameters, such as the decision tree structure, training iterations, fixed configuration or prediction multiplier, etc. We evaluate our system for a variety of expected accuracy targets, explained next.

To compare the performance of the baseline and proposed approaches, we perform controlled experiments to keep the accuracy achieved by the two approaches at the same level in order to compare the average latency numbers. Given an accuracy target, we perform binary search to find the minimum fixed configuration for the baseline and minimum prediction *multiplier* (as described in Section 5.1.2) for the proposed approach to reach this desired accuracy. Then we compare the average latency numbers at each accuracy target. Prediction overhead is included in the end-to-end latency. For the accuracy target, we use recall-at-1 (the fraction of queries where the top-1 nearest neighbor returned from search is (one of) the ground truth nearest neighbor) for the cases without compression. For the cases with compression, we use recall-at-100 (the fraction of queries where the top-100 nearest neighbors returned from search include (one of) the ground truth nearest neighbor) as the accuracy target since it's challenging for compression-based approaches to reach high recall-at-1: the vector compression introduces distance precision loss which could reorder the ranks of nearest neighbors. We search and return top-1 or top-100 nearest neighbors corresponding to the recall-at-1/at-100 metrics. We process the queries one by one without batching by default. And we measure the average latency in single-thread as in previous work [57, 79, 80].

5.2.2 IVF without compression

Approach 1: decision trees, single prediction

Figure 5.4 plots the average end-to-end latency vs. recall-at-1, comparing fixed configuration and adaptive prediction. Table 5.7 presents the corresponding detailed numbers. Overall, our approach provides consistent latency reduction from 13% to 58% compared to fixed configurations at recall-at-1 targets between 0.95 and 1. The relative latency reduction increases as recall target gets higher. This is because the baseline approach needs to use a larger fixed configuration to reach a higher recall target, which gives our approach



(a) DEEP10M

(b) SIFT10M

(c) GIST1M

Figure 5.4: IVF index: Average end-to-end latency vs. recall-at-1. Note the y-axis starts at 0.95. The yellow line is a simple heuristic approach for comparison.

more room to improve. GIST has higher average latency due to the two reasons described in Section 4.2.1.

DEEP10M	Fixed	Adaptive	
Recall-at-1	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	2.015 ms	1.743 ms	13%
0.96	2.390 ms	1.903 ms	20%
0.97	2.857 ms	2.110 ms	26%
0.98	3.773 ms	2.496 ms	34%
0.99	5.547 ms	3.343 ms	40%
1.00	48.457 ms	21.315 ms	56%
SIFT10M	Fixed	Adaptive	
Recall-at-1	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	3.474 ms	2.492 ms	28%
0.96	3.980 ms	2.776 ms	30%
0.97	4.953 ms	3.232 ms	35%
0.98	6.201 ms	3.819 ms	38%
0.99	8.376 ms	5.138 ms	39%
1.00	43.304 ms	20.639 ms	52%
GIST1M	Fixed	Adaptive	
Recall-at-1	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	16.142 ms	12.108 ms	25%
0.96	17.837 ms	14.544 ms	18%
0.97	19.981 ms	15.656 ms	22%
0.98	22.948 ms	17.576 ms	23%
0.99	38.068 ms	22.654 ms	40%
1.00	70.959 ms	29.875 ms	58%

Table 5.7: IVF index: Average end-to-end latency at different recall-at-1 targets.

Figure 5.4 also includes the performance of a simple heuristic-based approach: for each query, we search all the clusters whose centroid-to-query distances are within $x\%$ (e.g., 140%) of the shortest centroid-to-query distance, and we apply different x to reach different recall targets. Results show that this heuristic approach is only able to provide some improvement at a few cases. Since the baseline HNSW index already employs a beam search heuristic as explained in Section 4.1.2, we do not present another heuristic for HNSW.

DEEP10M Recall-at-1 Target	Fixed Configuration Avg. Latency	Single Prediction After 6 Clusters Avg. Latency (Reduction)	Multi-Prediction After 6,11 Clusters Avg. Latency (Reduction)	Multi-Prediction After 3,6,11 Clusters Avg. Latency (Reduction)
0.95	2.015 ms	1.743 ms (13%)	1.754 ms (13%)	1.727 ms (14%)
0.96	2.390 ms	1.903 ms (20%)	1.900 ms (21%)	1.880 ms (21%)
0.97	2.857 ms	2.110 ms (26%)	2.086 ms (27%)	2.076 ms (27%)
0.98	3.773 ms	2.496 ms (34%)	2.444 ms (35%)	2.439 ms (35%)
0.99	5.547 ms	3.343 ms (40%)	3.186 ms (43%)	3.241 ms (42%)
1.00	48.457 ms	21.315 ms (56%)	19.421 ms (60%)	19.365 ms (60%)

Table 5.8: IVF index: Average end-to-end latency at different recall-at-1 targets, with different numbers of predictions per query.

Approach 2: decision trees, multiple predictions

In the first approach, we make a single prediction after a fixed amount of search for each query. However, different intermediate search results provide different information, which motivates us to consider a second approach: making different number of predictions for each query. For a query whose nearest neighbor is easy to find, a single prediction is probably enough. However, for a difficult query we may want to perform multiple predictions at different timestamps to refine the prediction along the search. In addition to approach 1 where we make a single prediction after search top-6 clusters for every query, we evaluate the cases making multiple predictions for each query depending on the prediction value. How to choose the timestamps of each prediction is a grid search problem similar to the single prediction case.

Table 5.8 shows the results when making different number of predictions for the DEEP 10M dataset. For the multi-prediction case, we make up to 2 or 3 predictions after search different number of top clusters. If one prediction value is smaller than the next timestamp, the search will terminate without making another prediction. The results show that making multiple predictions for some queries could provide additional latency reduction, compared to always making a single prediction. On the other hand, it is a diminishing return to make even more predictions due to the nontrivial prediction overhead.

Approach 3: neural networks, single prediction

In the first two approaches we use GBDT decision tree models. We were curious whether different machine learning models could provide similar accuracy for accuracy-cost optimization problems. Thus we explore the third approach where we built simple neural net-

DEEP10M Recall-at-1 Target	Fixed Configuration Avg. Latency	Decision Trees Single Prediction Avg. Latency (Reduction)	Neural Networks Single Prediction Avg. Latency (Reduction)
0.95	2.015 ms	1.743 ms (13%)	1.998 ms (1%)
0.96	2.390 ms	1.903 ms (20%)	2.174 ms (9%)
0.97	2.857 ms	2.110 ms (26%)	2.391 ms (16%)
0.98	3.773 ms	2.496 ms (34%)	2.791 ms (26%)
0.99	5.547 ms	3.343 ms (40%)	3.695 ms (33%)
1.00	48.457 ms	21.315 ms (56%)	18.098 ms (63%)

Table 5.9: IVF index: Average end-to-end latency at different recall-at-1 targets, comparing decision trees and neural networks models.

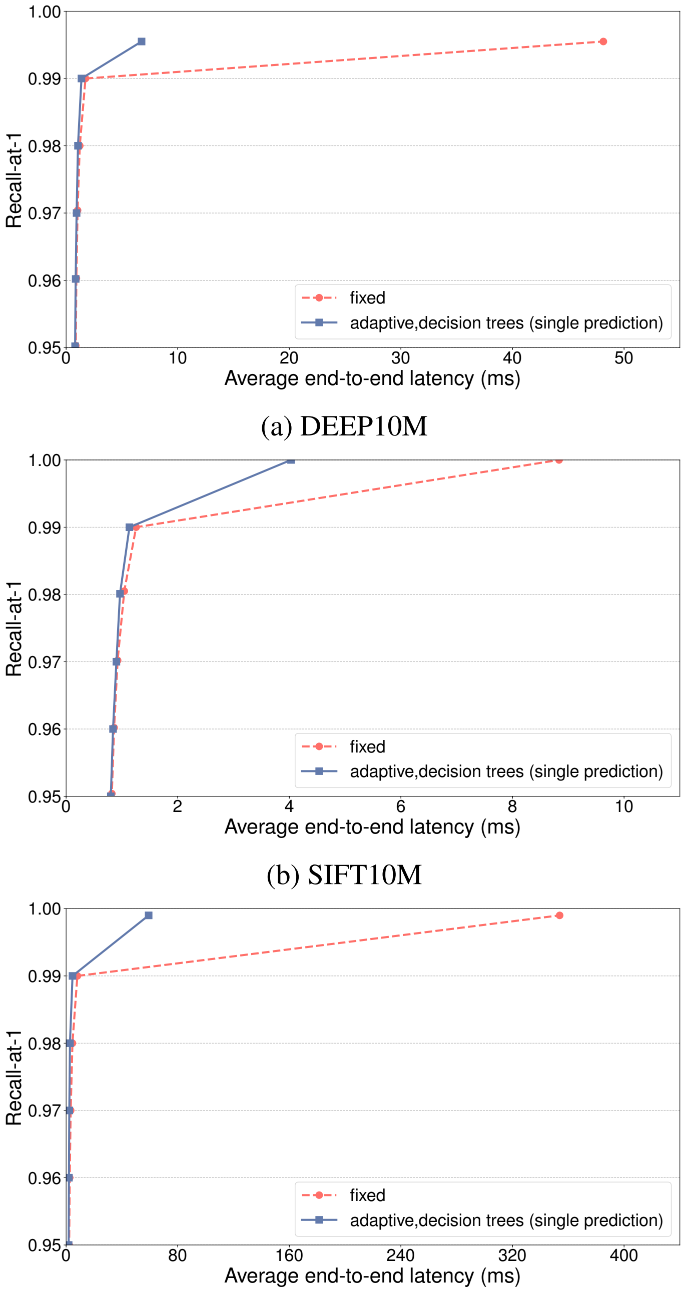
works model to predict the termination condition as described in Section 5.1.1. Table 5.9 shows the results comparing decision trees and neural networks for the DEEP 10M dataset. Compared with the decision trees model, the neural network approach provides better performance at the 1.00 accuracy target (even better than decision trees multi-predictions). On the other hand, neural networks approach has worse performance than decision trees at other accuracy targets, but is still able to provide latency reduction compared to the baseline. This is due to the higher prediction overhead of neural networks, which could be further alleviated as described in Section 5.2.1. And the most important takeaway is that the proposed learned termination idea does not depend on any specific machine learning model.

5.2.3 HNSW without compression

Approach 1: decision trees, single prediction

Figure 5.5 plots the average end-to-end latency vs. recall-at-1, comparing fixed configuration and adaptive prediction. Table 5.10 presents the corresponding detailed numbers. For DEEP and GIST we stop at 0.9955 and 0.999 recall target due to HNSW graph connectivity issue which make both approaches unable to find the nearest neighbor for a few queries. Overall, our approach provides consistent latency reduction from 2% to 86% compared to fixed configurations at recall-at-1 targets between 0.95 and 1.

The baseline has very high latency for some recall targets. This is because of the HNSW graph index structure. The performance of HNSW depends heavily on the distance between the query and the base layer start node, which acts like a dynamic cluster centroid compared to the static cluster centroids in IVF. When the start node is close to the query,



(a) DEEP10M

(b) SIFT10M

(c) GIST1M

Figure 5.5: HNSW index: Average end-to-end latency vs. recall-at-1. Note the y-axis starts at 0.95.

we can find the nearest neighbor very fast. As a result for most of the queries it takes shorter time to find the nearest neighbor in HNSW than IVF. On the other hand, the start

DEEP10M	Fixed	Adaptive	
Recall-at-1	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	0.865 ms	0.805 ms	7%
0.96	0.918 ms	0.856 ms	7%
0.97	1.027 ms	0.939 ms	9%
0.98	1.223 ms	1.063 ms	13%
0.99	1.737 ms	1.375 ms	21%
0.9955	48.145 ms	6.762 ms	86%

SIFT10M	Fixed	Adaptive	
Recall-at-1	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	0.819 ms	0.801 ms	2%
0.96	0.860 ms	0.841 ms	2%
0.97	0.923 ms	0.901 ms	2%
0.98	1.042 ms	0.966 ms	7%
0.99	1.255 ms	1.135 ms	10%
1.00	8.835 ms	4.032 ms	54%

GIST1M	Fixed	Adaptive	
Recall-at-1	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	2.185 ms	1.801 ms	18%
0.96	2.570 ms	2.014 ms	22%
0.97	3.269 ms	2.288 ms	30%
0.98	4.529 ms	2.758 ms	39%
0.99	8.064 ms	4.588 ms	43%
0.999	353.831 ms	59.162 ms	83%

Table 5.10: HNSW index: Average end-to-end latency at different recall-at-1 targets. For DEEP10M and GIST1M we stop at 0.9955 and 0.999 recall target due to HNSW graph connectivity issue.

node could be far away from the query due to rare graph connectivity issue or search difficulty issue as explained in the end of Section 4.2.1. In those rare cases HNSW would need much more distance evaluations than IVF, which forces the baseline approach to use an exceptionally large fixed configuration. Our approach could identify those rare cases and cover them with much lower average latency. This is why we can achieve a latency reduction up to 86%, which is a 7.1 times speedup.

DEEP10M Recall-at-1 Target	Fixed Configuration Avg. Latency	Single Prediction After 368 Eval. Avg. Latency (Reduction)	Multi-Prediction After 368,1003 Eval. Avg. Latency (Reduction)	Multi-Prediction After 265,368,1003 Eval. Avg. Latency (Reduction)
0.95	0.865 ms	0.805 ms (7%)	0.785 ms (9%)	0.800 ms (8%)
0.96	0.918 ms	0.856 ms (7%)	0.830 ms (10%)	0.846 ms (8%)
0.97	1.027 ms	0.939 ms (9%)	0.908 ms (12%)	0.921 ms (10%)
0.98	1.223 ms	1.063 ms (13%)	1.014 ms (17%)	1.015 ms (17%)
0.99	1.737 ms	1.375 ms (21%)	1.320 ms (24%)	1.289 ms (26%)
0.9955	48.145 ms	6.762 ms (86%)	5.097 ms (89%)	6.425 ms (87%)

Table 5.11: HNSW index: Average end-to-end latency at different recall-at-1 targets, with different numbers of predictions (after different number of distance evaluations) per query.

Approach 2: decision trees, multiple predictions

Similar to the IVF index case, we evaluate the cases making multiple predictions for each query depending on the prediction value. Table 5.11 shows the results when making different number of predictions for the DEEP 10M dataset. For the multi-prediction case, we make up to 2 or 3 predictions after different number of distance evaluations. The results show that making multiple predictions for some queries could provide additional latency reduction, compared to always making a single prediction. On the other hand, it is again a diminishing return to make even more predictions due to the nontrivial prediction overhead.

Approach 3: neural networks, single prediction

Similar to the IVF index case, we explore the third approach where we built simple neural networks model to predict the termination condition. Table 5.12 shows the results comparing decision trees and neural networks for the DEEP 10M dataset. Compared with the decision trees model, the neural network approach provides better performance at the 1.00 accuracy target (close to the performance of decision trees multi-predictions). On the other hand, neural networks approach has worse performance than decision trees at other accuracy targets, even worse than the baseline for certain cases. This is due to the higher prediction overhead of neural networks and due to the high efficiency of the HNSW index at lower accuracy targets. However, neural networks prediction overhead could be optimized, and again it proves that the proposed learned termination idea does not depend on any specific machine learning model.

DEEP10M Recall-at-1 Target	Fixed Configuration Avg. Latency	Decision Trees Single Prediction Avg. Latency (Reduction)	Neural Networks Single Prediction Avg. Latency (Reduction)
0.95	0.865 ms	0.805 ms (7%)	1.119 ms (-29%)
0.96	0.918 ms	0.856 ms (7%)	1.170 ms (-27%)
0.97	1.027 ms	0.939 ms (9%)	1.243 ms (-21%)
0.98	1.223 ms	1.063 ms (13%)	1.373 ms (-12%)
0.99	1.737 ms	1.375 ms (21%)	1.721 ms (1%)
0.9955	48.145 ms	6.762 ms (86%)	5.189 ms (89%)

Table 5.12: HNSW index: Average end-to-end latency at different recall-at-1 targets, comparing decision trees and neural networks models.

5.2.4 IVF with OPQ compression (decision trees, single prediction)

Table 5.13 presents the results when applying our approach to IVF index with OPQ compression, which is one of the stat-of-the-art vector quantization methods [42]. We use OPQ to transform the vectors with a compression factor of 8 (i.e., OPQ48 for DEEP, OPQ64 for SIFT, OPQ480 for GIST). One thing to note is that the results in Table 5.13 are not directly comparable to the results in Table 5.7 because the index construction, the memory overhead, and the recall target definition are different.

Overall, our approach provides consistent latency reduction from 1% to 52% compared to fixed configurations at recall-at-100 targets between 0.95 and 1. Our approach has less improvement for GIST due to the distance precision loss: GIST has higher number of dimensions than DEEP and SIFT; With the same compression factor, larger number of dimensions lead to larger absolute precision loss by compression; This affects the precision of intermediate search results features, which leads to lower prediction accuracy. Nevertheless, the results show that the proposed approach is effective when vector compression is applied.

5.2.5 IMI with OPQ compression (decision trees, single prediction)

Table 5.14 presents the results when applying our approach to billion-scale datasets. We choose IMI index with OPQ compression as the baseline, which is one of the state-of-the-art approaches for billion-scale ANN search [8]. As explained in Section 4.1.2, IMI index is a variant of IVF index so that we are able to apply the same approach to train the prediction model. We build IMI index with *(2^14)^2 = 268435456* clusters. Since the database is much larger, we increase the number of training iterations from 100 to

DEEP10M	Fixed	Adaptive	
Recall-at-100	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	2.446 ms	2.134 ms	13%
0.96	2.740 ms	2.318 ms	15%
0.97	3.223 ms	2.566 ms	20%
0.98	4.185 ms	3.006 ms	28%
0.99	5.952 ms	3.880 ms	35%
1.00	52.382 ms	25.284 ms	52%
SIFT10M	Fixed	Adaptive	
Recall-at-100	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	3.885 ms	2.983 ms	23%
0.96	4.276 ms	3.259 ms	24%
0.97	5.308 ms	3.731 ms	30%
0.98	6.721 ms	4.413 ms	34%
0.99	8.752 ms	5.744 ms	34%
1.00	49.732 ms	24.101 ms	52%
GIST1M	Fixed	Adaptive	
Recall-at-100	Configuration	Prediction	
Target	Avg. Latency	Avg. Latency	Reduction
0.95	36.792 ms	35.597 ms	3%
0.96	39.629 ms	39.263 ms	1%
0.97	44.299 ms	42.504 ms	4%
0.98	53.641 ms	52.865 ms	1%
0.99	84.044 ms	67.840 ms	19%
1.00	154.962 ms	111.186 ms	28%

Table 5.13: IVF index with OPQ compression: Average end-to-end latency at different recall-at-100 targets.

500 and decrease the learning rate from 0.2 to 0.05 to improve the accuracy. We stop at 0.995 recall target because it takes too long to reach 1.0 recall for billion-scale database. Overall, our approach provides consistent latency reduction from 8% to 59% compared to fixed configurations at recall-at-100 targets between 0.95 and 0.995.

DEEP1B Recall-at-100 Target	Fixed Configuration Avg. Latency	Adaptive Prediction Avg. Latency	Reduction
0.95	39.994 ms	36.911 ms	8%
0.96	52.353 ms	41.386 ms	21%
0.97	70.287 ms	48.398 ms	31%
0.98	97.558 ms	58.907 ms	40%
0.99	166.346 ms	84.936 ms	49%
0.995	288.611 ms	117.920 ms	59%
SIFT1B Recall-at-100 Target	Fixed Configuration Avg. Latency	Adaptive Prediction Avg. Latency	Reduction
0.95	48.217 ms	34.215 ms	29%
0.96	58.051 ms	39.120 ms	33%
0.97	72.990 ms	45.692 ms	37%
0.98	100.894 ms	55.502 ms	45%
0.99	161.553 ms	81.423 ms	50%
0.995	257.333 ms	116.777 ms	55%

Table 5.14: IMI index with OPQ compression: Average end-to-end latency at different recall-at-100 targets.

5.2.6 Effect of batching (decision trees, single prediction)

In many of the latency-sensitive online serving scenarios we target, requests are often processed one by one as they arrive in order to make real-time response. But sometimes a small batch is also desirable. On the other hand, in offline analysis scenarios, requests are often combined in a single large batch to maximize the throughput. We set batch size = 1 (no batching) as default in previous sections. Table 5.15 presents the results with different batch sizes for DEEP10M dataset with IVF and HNSW indices without compression. We find that for both IVF and HNSW indices batching amortize some fixed computation or memory allocation cost across queries, but the amount of latency reduction from the proposed approach stays similar.

For IVF, it is faster to compute the distance between cluster centroids and a batch of queries because Faiss switches from CPU SIMD vectorization to drastically more batch-efficient BLAS matrix-matrix operations when the batch size reaches 20. Thus using a different batch size is equivalent to adding/subtracting similar amount of latency from both the baseline and our approach.

IVF index Recall-at-1 Target	Fixed Configuration Avg. Latency Batch=1/100/10000	Adaptive Prediction Avg. Latency Batch=1/100/10000
0.95	2.015/1.904/1.894 ms	1.743/1.665/1.654 ms
0.96	2.390/2.274/2.266 ms	1.903/1.823/1.814 ms
0.97	2.857/2.729/2.717 ms	2.110/2.024/2.011 ms
0.98	3.773/3.633/3.620 ms	2.496/2.411/2.402 ms
0.99	5.547/5.371/5.370 ms	3.343/3.244/3.243 ms
1.00	48.457/48.164/48.198 ms	21.315/21.264/21.157 ms
HNSW index Recall-at-1 Target	Fixed Configuration Avg. Latency Batch=1/100/10000	Adaptive Prediction Avg. Latency Batch=1/100/10000
0.95	0.865/0.457/0.417 ms	0.805/0.387/0.369 ms
0.96	0.918/0.515/0.474 ms	0.856/0.443/0.419 ms
0.97	1.027/0.628/0.581 ms	0.939/0.523/0.498 ms
0.98	1.223/0.839/0.774 ms	1.063/0.651/0.617 ms
0.99	1.737/1.370/1.270 ms	1.375/0.952/0.912 ms
0.9955	48.145/47.732/47.300 ms	6.762/6.225/6.018 ms

Table 5.15: DEEP10M without compression: Average end-to-end latency at different recall-at-1 targets with different batch sizes. Batch size = 1 (no batching) is used in all the previous experiments.

For HNSW, an array of size n (the number of database vectors) is allocated every time the queries are sent to the database. This array is used to record which database vectors have been visited for each query, since HNSW's graph traversal may reach the same node multiple times. When batching is enabled, the array is shared by multiple queries and the memory allocation cost is amortized.

5.3 Related Work

In addition to the related work mentioned in Section 4.1, there are many recent works about ANN search in both database and machine learning communities. In database communities several works focused on improving the Locality Sensitive Hashing (LSH) technique: Data Sensitive Hashing improves the hashing functions and hashing family based on the data distributions [40]. Neighbor-Sensitive Hashing improves approximate kNN search based on an unconventional observation that magnifying the Hamming distances among neighbors helps in their accurate retrieval [88]. LazyLSH uses a single base index to support the computations in multiple metric spaces, significantly reducing the maintenance overhead [112].

In machine learning communities several works focused on improving the vector compression technique: SUBIC uses deep convolutional neural networks to produce supervised, compact, structured binary codes for visual search [56]. Wu *et al*. proposed an end-to-end trainable multiscale quantization method that minimizes overall quantization loss [106].

Several works focused on early stopping conditions for exact nearest neighbor search. Ciaccia *et al*. proposed probabilistic early stopping conditions for exact NN search in high-dimensional and complex metric spaces on smaller 12K to 100K datasets [26]. Gogolou *et al*. presented ideas on how to provide probabilistic estimates of the final answer to help users decide when to stop an exact NN search query on 100M to 267M datasets [43].

The proposed adaptive early termination technique deals with a similar problem in the online/progressive query answering communities. Online query answering relies on user interactions to iteratively refine the query results [47], which is similar to the proposed adaptive search termination that leverages machine learning models to predict the quality of intermediate search results. Recently, Turkay *et al*. proposed a cognitive model of human-computer interaction as the underlying mechanism to determine the pace of user interaction for high-dimensional data analysis such as online PCA and clustering [99]. Northstar [64] is another interactive data science system that uses interactive whiteboards to provide a highly collaborative visual data science environment.

5.4 Conclusion

Approximate nearest neighbor search algorithms aim to balance accuracy and cost (latency). We show, however, that traditional fixed configuration-based approaches lead to undesirably high average latency to reach high recall, because they fail to take into account the distribution of query difficulty. In this paper, we have demonstrated that there exist opportunities to exploit the variation in search termination conditions between queries. We have presented the first prediction-based approach to leverage this inter-query variation and improve end-to-end performance, substantially reducing average latency. We believe that the practicality and effectiveness of this approach make it a must-use component for the approximate nearest neighbor toolkit.

Chapter 6

Conclusion and Future Work

In this dissertation, we demonstrated how modular system approaches can be used to balance accuracy and computation cost for recommendation systems, and how lightweight machine learning models can be used to adaptively answer "how much (more) work should be done for this query?" inside the systems. First we presented a high-level caching system above the scoring component to avoid unnecessary expensive computations. The heuristic-based cache saves operating costs on queries with low revenue in history. In addition, we showed that lightweight machine learning models (GBDT) can provide more accurate per-query cache refresh decisions than heuristics, which lead to a better balance between ads revenue expectations and computation cost. This provides a potential hundred million dollar net profit gain in a quarter for the Bing advertising system.

Since caching introduces different penalties for each query, it is not always possible to avoid all computations (e.g., queries with high revenue expectation in search advertising). We therefore also focused on applying our core technique to a key component within these systems. We introduced adaptive early termination for approximate nearest neighbor search inside the candidate retrieval component. We showed that lightweight machine learning models (GBDT and neural networks) can provide adaptive per-query search termination conditions, which lead to a better balance between search accuracy and latency. This advanced the Pareto frontier in terms of nearest neighbor search accuracy and latency compared with state-of-the-art ANN approaches.

Of the many potential generalizations and extensions of the ideas presented in this dissertation, we discuss two that best help place our work in a broader context:

6.1 More applications for adaptive caching and early termination

In this dissertation we focused on designing adaptive caching systems for search advertising systems to balance the cached results quality and operating cost. However, our findings can also be applied to caching systems above other machine learning systems. For example, in Section 2.2 we mentioned about caching for general web search, where previous related works build caching heuristics. Different heuristics take a subset of the features into consideration: recency, frequency, processing cost, and ranking score of the query and so on. It would be interesting to apply the state-of-the-art machine learning models to build a prediction-based web search cache to provide a better balance between web search quality and processing cost. In addition, prediction-based cache design can also be applied to areas like content-based image retrieval [33, 34] and question answering systems [30] where the query results are approximate and expensive.

To apply our core technique to the components below the caching system, we focused on designing adaptive early termination for approximate nearest neighbor search queries to balance the search accuracy and latency. However, our findings can also be applied to other information retrieval query processing problems where queries require various processing costs. For example, image retrieval [111], spoken queries processing [105], and document retrieval [78] all have the concepts of query early termination. It would be interesting to apply prediction-based early termination to balance the query result quality and processing costs in those scenarios.

6.2 Adaptive machine learning-based decision making for computer systems in general

The work presented herein a specific case of a broader question increasingly being discussed at the intersection of systems and machine learning: How to apply lightweight machine learning models to replace heuristics or similar decisions in real-world systems. As discussed in Jeff Dean's SysML conference 2018 talk [29] and in the MLSys whitepaper [92], computer systems are filled with heuristics. These heuristics are manually tuned to work well in "general cases", but they generally do not adapt to patterns of use and do not take into account available context. For example, in the caching work we showed that a traditional cache with a fixed refresh rate would lead to high revenue loss. Even the precisely tuned heuristics we propose are not able to maximize the benefit of caching. In the ANN search work, we showed that using the fixed search termination configurations would lead to inefficiency and high average latency when trying to reach high accuracy.

Compare to tuning heuristics, it is much easier to train a machine learning model to take the actual pattern and many other context features into consideration. It would be interesting to consider machine learning alternatives anywhere we use heuristics to make a decision: indexing [65], database tuning [76, 90, 100], network traffic scheduling [71, 74], and so on. In the introduction we mentioned that there exist exceptions where modular system approaches are designed to work with approximate and noisy results. For instance, in approximate database query processing [84] only a small fraction of the relevant tuples is processed in order to provide fast, approximate answers. This is another example where machine learning can shine by learning from past queries to answer new queries using increasingly smaller portions of data.

In the caching work, we showed that a machine learning model can easily handle richer set of features and generate more accurate cache refresh decisions, which leads to greater net profit gain by caching. In the ANN search work, we showed that a machine learning model can incorporate different kinds of features (query vector patterns and intermediate result contexts) and generate per-query adaptive search termination condition, which provides a better balance between accuracy and latency. Our work helps move the question about "how to apply lightweight machine learning models to replace heuristics" forward in two ways. First, it provides confirmation that these ideas can be made to work in specific contexts, and help further motivate the search for even more-general techniques that can be widely applied. Second, it demonstrates how these techniques can be applied to important systems where it was not previously clear that a learned heuristic could be effectively and generally applied to improve the cost/accuracy tradeoff.

Bibliography

[1] Average CTR (Click-Through Rate): Learn How Your CTR Compares. `http://www.wordstream.com/average-ctr`, 2016. 2.3.2, 3, 3.2.1

[2] Microsoft Earnings Release FY16 Q4. `https://www.microsoft.com/en-us/Investor/earnings/FY-2016-Q4/`, 2017. 3.3.2

[3] SPTAG: A library for fast approximate nearest neighbor search. `https://github.com/Microsoft/SPTAG`, 2018. 4, 4.1.2

[4] LightGBM Documentation. `https://lightgbm.readthedocs.io/en/latest/index.html`, 2020. 5.1.1

[5] Sadiye Alici, Ismail Sengor Altingovde, Rifat Ozcan, Berkant Barla Cambazoglu, and Özgür Ulusoy. Timestamp-based Result Cache Invalidation for Web Search Engines. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 973–982, 2011. 2.2.1

[6] Sadiye Alici, Ismail Sengor Altingovde, Rifat Ozcan, B. Barla Cambazoglu, and Özgür Ulusoy. Adaptive Time-to-Live Strategies for Query Result Caching in Web Search Engines. In *Proceedings of the 34th European Conference on Information Retrieval (ECIR)*, pages 401–412, 2012. 2.2.1, 3.1.4

[7] Artem Babenko and Victor Lempitsky. Additive Quantization for Extreme Vector Compression. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 931–938, 2014. 4

[8] Artem Babenko and Victor Lempitsky. The Inverted Multi-Index. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(6):1247–1260, 2015. 4, 4.1.2, 4.1.2, 5, 5.2.5

[9] Artem Babenko and Victor Lempitsky. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2055–2063, 2016. 4.1, 4.2.1, 5

[10] Ricardo Baeza-Yates, Aristides Gionis, Flavio P Junqueira, Vanessa Murdock, Vassilis Plachouras, and Fabrizio Silvestri. Design Trade-Offs for Search Engine

Caching. *ACM Transactions on the Web (TWEB)*, 2(4):1–28, 2008. 2.2.1

[11] Xiao Bai and Flavio P. Junqueira. Online Result Cache Invalidation for Real-time Web Search. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 641–650, 2012. 2.2.1

[12] Dmitry Baranchuk, Artem Babenko, and Yury Malkov. Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors. In *European Conference on Computer Vision (ECCV)*, pages 202–216, 2018. 4, 4.1.2

[13] Konstantin Berlin, Sergey Koren, Chen-Shan Chin, James P. Drake, Jane M. Landolin, and Adam M. Phillippy. Assembling Large Genomes with Single-Molecule Sequencing and Locality-Sensitive Hashing. *Nature Biotechnology*, 33(6):623–630, 2015. 4

[14] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When Is "Nearest Neighbor" Meaningful? In *International Conference on Database Theory (ICDT)*, pages 217–235, 1999. 4

[15] Christian Böhm, Stefan Berchtold, and Daniel A. Keim. Searching in High-Dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases. *ACM Computing Surveys*, 33(3):322–373, 2001. 4

[16] Edward Bortnikov, Ronny Lempel, and Kolman Vornovitsky. Caching for Realtime Search. In *Proceedings of the 33rd European Conference on Information Retrieval (ECIR)*, pages 104–116, 2011. 2.2.1, 3.1.4

[17] Léon Bottou. Large-Scale Machine Learning with Stochastic Gradient Descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010. 5.1.1, 5.1.1

[18] B Barla Cambazoglu and Ismail Sengor Altingovde. Impact of Regionalization on Performance of Web Search Engine Result Caches. In *Proceedings of the 19th Symposium on String Processing and Information Retrieval (SPIRE)*, pages 161–166, 2012. 2.2.1, 3.1.1

[19] Berkant Barla Cambazoglu, Flavio P. Junqueira, Vassilis Plachouras, Scott Banachowski, Baoqiu Cui, Swee Lim, and Bill Bridge. A Refreshing Perspective of Search Engine Caching. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, pages 181–190, 2010. 2.2.1

[20] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G. Andersen, Michael Kaminsky, and Subramanya R. Dulloor. Scaling Video Analytics on Constrained Edge Nodes. In *Proceedings of the 2nd SysML Conference (SysML)*, 2019. 5.1.1

[21] Pei Cao and Sandy Irani. Cost-aware WWW Proxy Caching Algorithms. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pages 193–206, 1997. 3.1.1

[22] Kaushik Chakrabarti and Sharad Mehrotra. The Hybrid Tree: An Index Structure for High Dimensional Feature Spaces. In *International Conference on Data Engineering (ICDE)*, pages 440–447, 1999. 4.1.2

[23] Moses S. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 380–388, 2002. 4, 4.1.1

[24] Yongjian Chen, Tao Guan, and Cheng Wang. Approximate Nearest Neighbor Search by Residual Vector Quantization. *Sensors*, 10(12):11259–11273, 2010. 4

[25] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, et al. Serving DNNs in Real Time at Datacenter Scale with Project Brainwave. *IEEE Micro*, 38(2):8–20, 2018. 5.1.1

[26] Paolo Ciaccia and Marco Patella. PAC Nearest Neighbor Queries: Approximate and Controlled Search in High-Dimensional and Metric Spaces. In *International Conference on Data Engineering (ICDE)*, pages 244–255, 2000. 5.3

[27] Abhinandan S. Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. Google News Personalization: Scalable Online Collaborative Filtering. In *Proceedings of the 16th International Conference on World Wide Web (WWW)*, pages 271–280, 2007. 1, 4

[28] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In *Proceedings of the 20th Annual Symposium on Computational Geometry (SCG)*, pages 253–262, 2004. 4.1.2

[29] Jeff Dean. Systems and Machine Learning Symbiosis. `https://youtu.be/Nj6uxDki6-0`, 2018. 6.2

[30] David Dominguez-Sal, Josep Lluis Larriba-Pey, and Mihai Surdeanu. A multi-layer collaborative cache for question answering. In *European Conference on Parallel Processing*, pages 295–306, 2007. 6.1

[31] Matthijs Douze, Alexandre Sablayrolles, and Hervé Jégou. Link and Code: Fast Indexing With Graphs and Compact Regression Codes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3646–3654, 2018. 4, 4.1.2

[32] Tiziano Fagni, Raffaele Perego, Fabrizio Silvestri, and Salvatore Orlando. Boosting the Performance of Web Search Engines: Caching and Prefetching Query Results by Exploiting Historical Usage Data. *ACM Transactions on Information Systems (TOIS)*, 24(1):51–78, 2006. 2.2.1, 3.1.4

[33] Fabrizio Falchi, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fausto Rabitti. Caching content-based queries for robust and efficient image retrieval. In

Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, pages 780–790, 2009. 6.1

[34] Fabrizio Falchi, Claudio Lucchese, Salvatore Orlando, Raffaele Perego, and Fausto Rabitti. Similarity caching in large-scale image retrieval. *Information processing & management*, 48(5):803–818, 2012. 6.1

[35] Jerome H. Friedman. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics*, 29(5):1189–1232, 2001. 1.1, 3, 3.2, 5, 5.1.1

[36] Cong Fu and Deng Cai. EFANNA : An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph. *arXiv preprint arXiv:1609.07228*, 2016. 4.1.2

[37] 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. 4, 4.1.2

[38] João Gama and Pavel Brazdil. Cascade generalization. *Machine Learning*, 41(3): 315–343, 2000. 1

[39] Qingqing Gan and Torsten Suel. Improved Techniques for Result Caching in Web Search Engines. In *Proceedings of the 18th International Conference on World Wide Web (WWW)*, pages 431–440, 2009. 2.2.1

[40] Jinyang Gao, Hosagrahar Visvesvaraya Jagadish, Wei Lu, and Beng Chin Ooi. DSH: Data Sensitive Hashing for High-Dimensional k-NN Search. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1127–1138, 2014. 5.3

[41] Jinyang Gao, H.V. Jagadish, Beng Chin Ooi, and Sheng Wang. Selective Hashing: Closing the Gap between Radius Search and k-NN Search. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 349–358, 2015. 4.1.2

[42] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. Optimized Product Quantization for Approximate Nearest Neighbor Search. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2946–2953, 2013. 4.1.1, 5, 5.2, 5.2.4

[43] Anna Gogolou, Theophanis Tsandilas, Themis Palpanas, and Anastasia Bezerianos. Progressive Similarity Search on Time Series Data. In *International Workshop on Big Data Visual Exploration and Analytics (BigVis)*, pages 1–1, 2019. 5.3

[44] Yunchao Gong and Svetlana Lazebnik. Iterative Quantization: A Procrustean Approach to Learning Binary Codes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 817–824, 2011. 4

[45] Albert Gordo and Florent Perronnin. Asymmetric Distances for Binary Embeddings. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*,

pages 729–736, 2011. 4.1.1

[46] Thore Graepel, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. Web-Scale Bayesian Click-Through Rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 13–20, 2010. 1.1, 2, 2.1, 2.2.2

[47] Peter J. Haas and Joseph M. Hellerstein. Online Query Processing: A Tutorial. *ACM SIGMOD Record*, 30(2):623, 2001. 5.3

[48] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. EIE: Efficient Inference Engine on Compressed Deep Neural Network. In *Proceedings of the 43rd International Symposium on Computer Architecture (ISCA)*, pages 243–254, 2016. 5.1.1

[49] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015. 5.1.1

[50] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. Practical Lessons from Predicting Clicks on Ads at Facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pages 1–9, 2014. 1.1, 2, 2.1, 2.2.2

[51] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012. 5.1.1

[52] Johannes Hoffart, Stephan Seufert, Dat Ba Nguyen, Martin Theobald, and Gerhard Weikum. KORE: Keyphrase Overlap Relatedness for Entity Disambiguation. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management (CIKM)*, pages 545–554, 2012. 4

[53] 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. 4.1.2

[54] Piotr Indyk and Rajeev Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC)*, pages 604–613, 1998. 4.1.2

[55] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167*, 2015. 5.1.1

[56] Himalaya Jain, Joaquin Zepeda, Patrick Pérez, and Rémi Gribonval. SUBIC: A Supervised, Structured Binary Code for Image Search. In *IEEE International Conference on Computer Vision (ICCV)*, pages 833–842, 2017. 5.3

[57] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011. 4, 4.1.1, 4.1.2, 4.1.2, 4.2.1, 5, 5.2.1

[58] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. Searching in One Billion Vectors: Re-rank with Source Coding. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 861–864, 2011. 4.1, 4.2.1, 5

[59] Myeongjae Jeon, Saehoon Kim, Seung-won Hwang, Yuxiong He, Sameh Elnikety, Alan L Cox, and Scott Rixner. Predictive Parallelization: Taming Tail Latencies in Web Search. In *Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 253–262, 2014. 2.2.2

[60] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, pages 1–1, 2019. 4.2.1, 5, 5.2.1

[61] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *Proceedings of the 44th International Symposium on Computer Architecture (ISCA)*, pages 1–12, 2017. 5.1.1

[62] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems (NIPS)*, pages 3146–3154, 2017. 3.2.3, 3.3.1, 5, 5.1.1

[63] Saehoon Kim, Yuxiong He, Seung-won Hwang, Sameh Elnikety, and Seungjin Choi. Delayed-Dynamic-Selective (DDS) Prediction for Reducing Extreme Tail Latency in Web Search. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining (WSDM)*, pages 7–16, 2015. 2.2.2

[64] Tim Kraska. Northstar: An Interactive Data Science System. *Proceedings of the VLDB Endowment*, 11(12):2150–2164, 2018. 5.3

[65] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The Case for Learned Index Structures. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 489–504, 2018. 6.2

[66] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012. 5.1.1

[67] Brian Kulis and Kristen Grauman. Kernelized Locality-Sensitive Hashing for Scalable Image Search. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2130–2137, 2009. 4

[68] Yifan Lei, Qiang Huang, Mohan Kankanhalli, and Anthony Tung. Sublinear Time Nearest Neighbor Search over Generalized Weighted Space. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*, pages 3773–3781, 2019. 4.1.2

[69] Conglong Li and Alan L. Cox. GD-Wheel: A Cost-aware Replacement Policy for Key-value Stores. In *Proceedings of the Tenth European Conference on Computer Systems (EuroSys)*, pages 1–15, 2015. 3.1.1

[70] Conglong Li, David G. Andersen, Qiang Fu, Sameh Elnikety, and Yuxiong He. Workload Analysis and Caching Strategies for Search Advertising Systems. In *Proceedings of the 2017 Symposium on Cloud Computing (SoCC)*, pages 170–180, 2017. 1, 5.1.1

[71] Conglong Li, Matthew K. Mukerjee, David G. Andersen, Srinivasan Seshan, Michael Kaminsky, George Porter, and Alex C. Snoeren. Using Indirect Routing to Recover from Network Traffic Scheduling Estimation Error. In *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 13–24, 2017. 6.2

[72] Conglong Li, David G. Andersen, Qiang Fu, Sameh Elnikety, and Yuxiong He. Better Caching in Search Advertising Systems with Rapid Refresh Predictions. In *Proceedings of the 2018 World Wide Web Conference (WWW)*, pages 1875–1884, 2018. 2, 5.1.1

[73] Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2020. 3

[74] He Liu, Matthew K. Mukerjee, Conglong Li, Nicolas Feltman, George Papen, Stefan Savage, Srinivasan Seshan, Geoffrey M. Voelker, David G. Andersen, Michael

Kaminsky, George Porter, and Alex C. Snoeren. Scheduling Techniques for Hybrid Circuit/Packet Networks. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, pages 1–13, 2015. 6.2

[75] Qin Lv, Moses Charikar, and Kai Li. Image Similarity Search with Compact Data Structures. In *Proceedings of the 13th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 208–217, 2004. 4, 4.1.1

[76] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J. Gordon. Query-Based Workload Forecasting for Self-Driving Database Management Systems. In *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 631–645, 2018. 6.2

[77] Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. Learning to Predict Response Times for Online Query Scheduling. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 621–630, 2012. 2.2.2

[78] Joel Mackenzie, Falk Scholer, and J Shane Culpepper. Early termination heuristics for score-at-a-time index traversal. In *Proceedings of the 22nd Australasian Document Computing Symposium*, pages 1–8, 2017. 6.1

[79] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate Nearest Neighbor Algorithm Based on Navigable Small World Graphs. *Information Systems*, 45:61–68, 2014. 4, 4.1.2, 4.1.2, 5.2.1

[80] 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*, 42(4):824–836, 2020. 4, 4.1.2, 4.1.2, 4.2.1, 5, 5.2.1

[81] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval.* Cambridge University Press, 2008. 4.1.2

[82] H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young, Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov, Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg, Arnar Mar Hrafnkelsson, Tom Boulos, and Jeremy Kubica. Ad Click Prediction: A View from the Trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 1222–1230, 2013. 1.1, 2, 2.1, 2.2.2

[83] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černockỳ, and Sanjeev Khudanpur. Recurrent Neural Network Based Language Model. In *11th Annual Conference of the International Speech Communication Association (INTERSPEECH)*, pages 1045–1048, 2010. 5.1.1

[84] Barzan Mozafari. Approximate query engines: Commercial challenges and research opportunities. In *Proceedings of the 2017 ACM International Conference on*

Management of Data, pages 521–524, 2017. 6.2

[85] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814, 2010. 5.1.1

[86] Rifat Ozcan, Ismail Sengor Altingovde, and Özgür Ulusoy. Cost-Aware Strategies for Query Result Caching in Web Search Engines. *ACM Transactions on the Web (TWEB)*, 5(2):1–25, 2011. 2.2.1

[87] Rifat Ozcan, Ismail Sengor Altingovde, B. Barla Cambazoglu, and Özgür Ulusoy. Second Chance: A Hybrid Approach for Dynamic Result Caching and Prefetching in Search Engines. *ACM Transactions on the Web (TWEB)*, 8(1):1–22, 2013. 2.2.1, 3.1.4

[88] Yongjoo Park, Michael Cafarella, and Barzan Mozafari. Neighbor-Sensitive Hashing. *Proceedings of the VLDB Endowment*, 9(3):144–155, 2015. 5.3

[89] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *Advances in Neural Information Processing Systems (NIPS) 2017 Autodiff Workshop*, 2017. 5, 5.1.1

[90] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. Self-Driving Database Management Systems. In *Conference on Innovative Data Systems Research (CIDR)*, 2017. 6.2

[91] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object Retrieval with Large Vocabularies and Fast Spatial Matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8, 2007. 4

[92] Alexander Ratner, Dan Alistarh, Gustavo Alonso, David G. Andersen, Peter Bailis, Sarah Bird, Nicholas Carlini, Bryan Catanzaro, Jennifer Chayes, Eric Chung, Bill Dally, Jeff Dean, Inderjit S. Dhillon, Alexandros Dimakis, Pradeep Dubey, Charles Elkan, Grigori Fursin, Gregory R. Ganger, Lise Getoor, Phillip B. Gibbons, Garth A. Gibson, Joseph E. Gonzalez, Justin Gottschlich, Song Han, Kim Hazelwood, Furong Huang, Martin Jaggi, Kevin Jamieson, Michael I. Jordan, Gauri Joshi, Rania Khalaf, Jason Knight, Jakub Konečný, Tim Kraska, Arun Kumar, Anastasios Kyrillidis, Aparna Lakshmiratan, Jing Li, Samuel Madden, H. Brendan McMahan, Erik Meijer, Ioannis Mitliagkas, Rajat Monga, Derek Murray, Kunle Olukotun, Dimitris Papailiopoulos, Gennady Pekhimenko, Theodoros Rekatsinas, Afshin Rostamizadeh, Christopher Ré, Christopher De Sa, Hanie Sedghi, Siddhartha Sen, Virginia Smith, Alex Smola, Dawn Song, Evan Sparks, Ion Stoica, Vivienne Sze, Madeleine Udell, Joaquin Vanschoren, Shivaram Venkataraman, Rashmi Vinayak, Markus Weimer, Andrew Gordon Wilson, Eric Xing, Matei Za-

haria, Ce Zhang, and Ameet Talwalkar. MLSys: The New Frontier of Machine Learning Systems. *arXiv preprint arXiv:1904.03257*, 2019. 6.2

[93] Francesco Ricci, Lior Rokach, and Bracha Shapira. Introduction to recommender systems handbook. In *Recommender Systems Handbook*, pages 1–35. Springer, 2011. 1

[94] Nick Roussopoulos, Stephen Kelley, and Frédéric Vincent. Nearest Neighbor Queries. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 71–79, 1995. 4.1

[95] Dennis W Ruck, Steven K Rogers, Matthew Kabrisky, Mark E Oxley, and Bruce W Suter. The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function. *IEEE Transactions on Neural Networks*, 1(4):296–298, 1990. 5

[96] Fethi Burak Sazoglu, B. Barla Cambazoglu, Rifat Ozcan, Ismail Sengor Altingovde, and Özgür Ulusoy. A Financial Cost Metric for Result Caching. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, pages 873–876, 2013. 2.2.1

[97] Josef Sivic and Andrew Zisserman. Video Google: A Text Retrieval Approach to Object Matching in Videos. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1470–1477, 2003. 4.1.2

[98] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 5.1.1

[99] Cagatay Turkay, Erdem Kaya, Selim Balcisoy, and Helwig Hauser. Designing Progressive and Interactive Analytics Processes for High-Dimensional Data Analysis. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):131–140, 2017. 5.3

[100] Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, and Bohan Zhang. Automatic Database Management System Tuning Through Large-Scale Machine Learning. In *Proceedings of the 2017 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 1009–1024, 2017. 6.2

[101] Jing Wang, Jingdong Wang, Gang Zeng, Zhuowen Tu, Rui Gan, and Shipeng Li. Scalable k-NN Graph Construction for Visual Descriptors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1106–1113, 2012. 4.1.2

[102] Jingdong Wang and Shipeng Li. Query-Driven Iterated Neighborhood Graph Search for Large Scale Indexing. In *Proceedings of the 20th ACM International Conference on Multimedia*, pages 179–188, 2012. 4.1.2

[103] Jingdong Wang, Naiyan Wang, You Jia, Jian Li, Gang Zeng, Hongbin Zha, and Xian-Sheng Hua. Trinary-Projection Trees for Approximate Nearest Neighbor

Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(2): 388–403, 2014. 4.1.2

[104] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24th VLDB Conference*, pages 194–205, 1998. 4, 4.1.1

[105] Jerome White, Douglas W Oard, Nitendra Rajput, and Marion Zalk. Simulating early-termination search for verbose spoken queries. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1270–1280, 2013. 6.1

[106] Xiang Wu, Ruiqi Guo, Ananda Theertha Suresh, Sanjiv Kumar, Daniel N. Holtmann-Rice, David Simcha, and Felix Yu. Multiscale Quantization for Fast Similarity Search. In *Advances in Neural Information Processing Systems (NIPS)*, pages 5745–5755, 2017. 5.3

[107] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical Evaluation of Rectified Activations in Convolution Network. *arXiv preprint arXiv:1505.00853*, 2015. 5.1.1

[108] Minjia Zhang and Yuxiong He. GRIP: Multi-Store Capacity-Optimized High-Performance Nearest Neighbor Search for Vector Search Engine. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM)*, pages 1673–1682, 2019. 4.1.2

[109] Minjia Zhang, Samyam Rajbhandari, Wenhan Wang, and Yuxiong He. DeepCPU: Serving RNN-based Deep Learning Models 10x Faster. In *2018 USENIX Annual Technical Conference (USENIX ATC)*, pages 951–965, 2018. 5.1.1

[110] Ting Zhang, Guo-Jun Qi, Jinhui Tang, and Jingdong Wang. Sparse Composite Quantization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4548–4556, 2015. 4

[111] Liang Zheng, Shengjin Wang, Ziqiong Liu, and Qi Tian. Fast image retrieval: Query pruning and early termination. *IEEE Transactions on Multimedia*, 17(5):648–659, 2015. 6.1

[112] Yuxin Zheng, Qi Guo, Anthony K.H. Tung, and Sai Wu. LazyLSH: Approximate Nearest Neighbor Search for Multiple Distance Functions with a Single Index. In *Proceedings of the 2016 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 2023–2037, 2016. 5.3

[113] Jingbo Zhou, Qi Guo, H.V. Jagadish, Lubos Krcal, Siyuan Liu, Wenhao Luan, Anthony K.H. Tung, Yueji Yang, and Yuxin Zheng. A Generic Inverted Index Framework for Similarity Search on the GPU. In *International Conference on Data Engineering (ICDE)*, pages 893–904, 2018. 4.1.2