WebReco: A Comprehensive Overview of an Industrial-Scale Webpage Recommendation System at Bing

Jaidev Shah, Iman Barjasteh, Amey Barapatre, Chuck Wang, Gang Luo, Rana Forsati
Jing Chu, Xugang Wang, Julie Fang, Fan Wu, Jiahui Liu, Xue Deng, Blake Shepard
Ronak Shah, Shaden Smith, Jon Soifer, Linjun Yang, Chuanjie Liu, Hongzhi Li, Rangan Majumder
{jaidevshah,abarapatre,imbarjas,charwan,gluo,raforsat,jingchu,xugwang}@microsoft.com
{juliefang,fwu,jiahui.liu,xuedeng,blakesh,rosh,shadensmith,jonso,linjya,chuanjieliu,hongzl,ranganm}@microsoft.com
Microsoft AI, USA

Abstract

In this paper, we introduce WebReco, a web-scale webpage recommendation system in Bing. Our product appears as *Explore Further* on the first page of Bing search results and is built to serve interesting, engaging, and complementary webpage recommendations alongside search results. We start by outlining our product north star, defining what it means to provide a good webpage recommendation for our product. We describe our evaluation methodology and provide a comprehensive overview of our production recommendation system, detailing each component from candidate generation to ranking. We dive into specifics of feature engineering and serving, model development and evolution, and how we scale our components to handle billions of webpages. We also share our approach to building a recommendation triggering model, a method for dynamically deciding the recommendation count given certain constraints. This paper provides a detailed look at building and optimizing a web-scale recommendation system, highlighting the design choices we made.

CCS Concepts

• **Information systems → Recommender systems**.

Keywords

Ranking, Recall, Multitask, Large Scale Recommender System, Large Language Models,

ACM Reference Format:
Jaidev Shah, Iman Barjasteh, Amey Barapatre, Chuck Wang, Gang Luo, Rana Forsati, Jing Chu, Xugang Wang, Julie Fang, Fan Wu, Jiahui Liu, Xue Deng, Blake Shepard, and Ronak Shah, Shaden Smith, Jon Soifer, Linjun Yang, Chuanjie Liu, Hongzhi Li, Rangan Majumder. 2024. WebReco: A Comprehensive Overview of an Industrial-Scale Webpage Recommendation System at Bing. In *Proceedings of 3rd International Workshop on Industrial Recommendation Systems (at CIKM 2024) (IRS '24).* ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

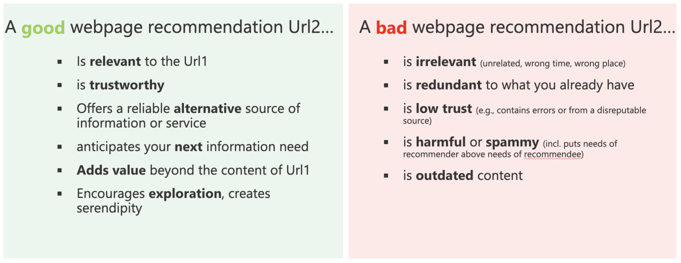
1 Introduction and Product Overview

Our product delivers query-agnostic webpage recommendations directly on the first page of Bing search results. Our goal is to enhance a Bing user's search experience by delivering engaging, complementary and exploratory webpage recommendations that are not neessarily tied to the narrow intent of the query. We have built, to our knowledge, one of the largest recommendation systems in the industry with a total index size of over 200 billion webpages. We serve billions of impressions a month across two major recommendation surfaces:

(1) **Inline Scenario:** Displayed below the top three search results, these recommendations appear as users view the initial search results page.
(2) **Clickback Scenario:** Shown when users return to the Bing search tab after clicking on a search result, providing additional webpage suggestions tied to the clicked URL.

We outline the north star of the product for the value we aim to bring to Bing users in Figure 1. In this paper, we'll use the term URL1 for the trigger webpage (the search result) and URL2 for a webpage recommendation. A good webpage recommendation is both trustworthy and provides additional value beyond the URL1 content. The very best recommendation experiences are created by presenting webpages that either encourage exploration of related topics, offer a reliable alternative perspective, or are capable of anticipating the next informational need of the user. Explicitly articulating what constitutes a good and a bad webpage recommendation was challenging to achieve consensus on, but was crucial. Having an explicit product north star enabled us to build RecoDCG, an LLM-powered recommendation quality metric with a prompt that incorporated the product goals into scoring instructions. LLMs give us the ability to build a metric that captures the north star of the product, thus motivating the team to productionize techniques that shift the product in the right direction.

In Figure 2, the jainar.net webpage about 'The Jain Diet' is the URL1 and the webpage recommendations under Explore Further are referred to as URL2s. For context, Jainism is a religion that originated in ancient India. In this example, our system generates a diverse set of recommendations: the "Jain Food Habits" webpage serves as an alternative resource, the "Jain Recipes" webpage anticipates a likely subsequent user intent for those interested in preparing meals that adhere to the dietary constraints, and the "Jain Dietary Practices: Unveiling Their Health Benefits" webpage offers users the opportunity to explore an additional aspect, specifically the health benefits of the diet.


A **good** webpage recommendation Url2...

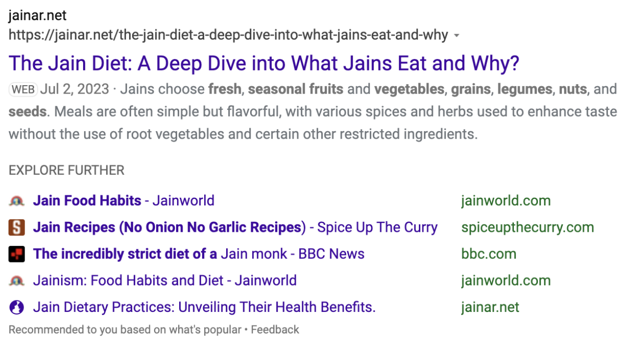
- Is **relevant** to the Url1
- is **trustworthy**
- Offers a reliable **alternative** source of information or service
- anticipates your **next** information need
- **Adds value** beyond the content of Url1
- Encourages **exploration**, creates serendipity

A **bad** webpage recommendation Url2...

- is **irrelevant** (unrelated, wrong time, wrong place)
- is **redundant** to what you already have
- is **low trust** (e.g., contains errors or from a disreputable source)
- is **harmful** or **spammy** (incl. puts needs of recommender above needs of recommendee)
- is **outdated** content

Figure 1: Product North Star for Web Recommendations

In this work, we provide a comprehensive overview of the development and evolution of our web-scale recommendation system, illustrated in Figure 3. We provide a deep dive of our end-to-end system, including metrics, the motivations behind our design choices, the models we deploy, feature engineering, and the data pipelines and mechanisms used for online feature serving.


jainar.net
https://jainar.net/the-jain-diet-a-deep-dive-into-what-jains-eat-and-why

The Jain Diet: A Deep Dive into What Jains Eat and Why?

WEB Jul 2, 2023 · Jains choose **fresh**, **seasonal fruits** and **vegetables**, **grains**, **legumes**, **nuts**, and **seeds**. Meals are often simple but flavorful, with various spices and herbs used to enhance taste without the use of root vegetables and certain other restricted ingredients.

EXPLORE FURTHER

Jain Food Habits - Jainworld	jainworld.com
Jain Recipes (No Onion No Garlic Recipes) - Spice Up The Curry	spiceupthecurry.com
The incredibly strict diet of a Jain monk - BBC News	bbc.com
Jainism: Food Habits and Diet - Jainworld	jainworld.com
Jain Dietary Practices: Unveiling Their Health Benefits.	jainar.net

Recommended to you based on what's popular • Feedback

Figure 2: Inline Scenario

2 Evaluation: Offline and Online Metrics

2.1 Offline: RecoDCG

Relevance and quality labels from human judges are expensive and slow to obtain. LLMs have been shown to be strong and consistent relevance judges, often outperforming human labelers[15]. For experimentation agility and to develop a quality signal, we developed a LLM-powered recommendation quality metric called RecoDCG to evaluate our production system and new techniques. At the heart of this metric is a pointwise scoring prompt that answers the question *how good of a recommendation is URL2 for a URL1*, reasoning over individual recommendation aspects (topicality, location, authority, and spam) before determining a recommendation quality score from 0 to 4. This prompt is tuned and evaluated against a human-judged URL1-URL2 quality evaluation set to ensure that the prompt accurately measures recommendation quality.

RecoDCG is based on the standard Discounted Cumulative Gain (DCG) formula, with the key distinction being the use of pointwise recommendation quality scores generated by a Large Language Model (LLM) as the relevance labels. As the RecoDCG measurement set, we use a diverse set of several thousand URL1s derived from historical queries from Bing users. This set is constructed to adequately represent a wide range of markets and languages. We scrape our production service for the top 5 ranked webpages for each URL1 in the evaluation set and calculate the RecoDCG@1, RecoDCG@3 and RecoDCG@5.

2.2 Offline: ClickNDCG

To facilitate rapid offline evaluation of click prediction models during the ranking stage, we utilize an offline metric termed ClickNDCG. This metric acts as an offline proxy for online click-through rate (CTR) and thus we refer to it as an inner-loop metric.

To collect logs that do not suffer from position or selection bias, we run a low-traffic online experiment where we randomly choose the top 5 URL2s from the ranking stage candidates to show to users, as well as the order. We retain logs with at least one *satisfied* (SAT) click, defined as a click where the user's information need was met. As a heuristic, we consider clicks with dwell times above 30 seconds to be SAT clicks. We calculate NDCG, assigning a relevance score of 100 for satisfied clicks (in order to make the metric easily interpretable) and 0 otherwise. Our experiments show that ClickNDCG closely aligns with our production system's online CTR, reflecting improvements or regressions accurately.

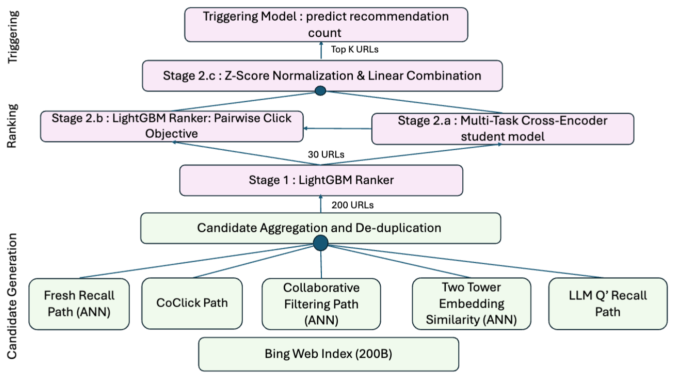

Triggering
Triggering Model : predict recommendation count
Top K URLs
Ranking
Stage 2.c : Z-Score Normalization & Linear Combination
Stage 2.b : LightGBM Ranker: Pairwise Click Objective
Stage 2.a : Multi-Task Cross-Encoder student model
30 URLs
Stage 1 : LightGBM Ranker
200 URLs
Candidate Generation
Candidate Aggregation and De-duplication
Fresh Recall Path (ANN)
CoClick Path
Collaborative Filtering Path (ANN)
Two Tower Embedding Similarity (ANN)
LLM Q' Recall Path
Bing Web Index (200B)

Figure 3: Production System Overview

2.3 Offline: Recall@k

To evaluate the Collaborative Filtering (CF) recall path shown in Figure 3, we report Recall@k. We sample a fixed set of URL1s that represent WebReco traffic and construct the evaluation set by collecting co-occurrence clicks (detailed in Section 3.1) from Bing session logs over a 1 week time frame after the training data window. A strong CF implementation will be able to generalize and obtain a Recall@k that handily beats the performance of the CoClick memorization path over the same CoClick graph. We use $k \in \{30, 100\}$ for reporting ablations.

2.4 Online: Page Click Through Rate (PTR)

$$PTR = \frac{\text{Impressions with at least 1 click on Explore Further}}{\text{Total Bing Impressions}}$$

2.5 Online: Conditional Click Through Rate (CTR)

$$CTR = \frac{\text{Impressions with at least 1 click on Explore Further}}{\text{Total Explore Further Impressions}}$$

3 Candidate Generation

Our production system has multiple candidate generators, referred to as recall paths. Each recall path is tuned to produce a specific number of candidates from Bing's 200 billion webpage index. These candidates are aggregated and de-duplicated before the ranking stage. The recall paths include:

3.1 CoClick (Co-occurrence Click) Recall Path

We define a CoClick as an event where multiple Bing user browsing sessions include clicks on both webpage A and webpage B. By this definition, CoClick counts the degree of co-occurrence in user sessions and implicitly represents the relatedness of the two webpages. Webpage B represents a common destination post query reformulations or shifts in user intent. Following the CoClick definition, we generate a CoClick graph across all webpages in the Bing index where the edge strength corresponds to the number of co-occurrence clicks over the considered time window. We store the CoClick counts in a performant key-value store where the key is the URL1 and the value is a list of top 30 URL2s based on CoClick counts. At serving time for a request corresponding to a specific URL1, our CoClick recall path generates candidates by returning the stored URL2 candidates.

The CoClick key-value store is updated with a weekly pipeline that processes billions of logs, taking a full day to run on a highly distributed compute cluster. The pipeline logic and pre-processing steps are detailed in the diagram in Figure 4.

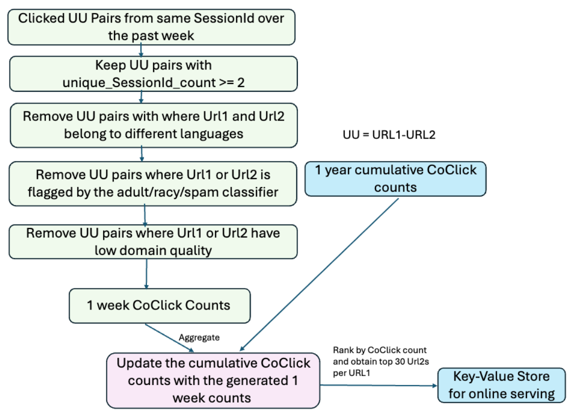

Clicked UU Pairs from same SessionId over the past week
Keep UU pairs with unique_SessionId_count >= 2
Remove UU pairs with where Url1 and Url2 belong to different languages
UU = URL1-URL2
Remove UU pairs where Url1 or Url2 is flagged by the adult/racy/spam classifier
1 year cumulative CoClick counts
Remove UU pairs where Url1 or Url2 have low domain quality
1 week CoClick Counts
Aggregate
Update the cumulative CoClick counts with the generated 1 week counts
Rank by CoClick count and obtain top 30 Url2s per URL1
Key-Value Store for online serving

Figure 4: CoClick Weekly Update Pipeline

3.2 Collaborative Filtering (CF) Recall Path

In this recall path, our recommendation system employs an item-to-item collaborative filtering (CF) approach. We train our CF model on the CoClick counts matrix constructed using over a year of Bing search logs. We employ Info-NCE contrastive loss [10] to optimize our embeddings. We conducted a comprehensive evaluation of various loss functions, including commonly used objectives such as Bayesian Personalized Ranking (BPR) [12] and binary hinge loss. Among these, the Info-NCE loss function, when carefully tuned, emerged as the superior choice, demonstrating the most effective performance across our experiments.

	$\tau = 0.01$	$\tau = 0.025$	$\tau = 0.07$	$\tau = 0.2$
$K = 5$	0.49	0.54	**0.63**	0.44
$K = 10$	0.50	0.55	0.58	0.43
$K = 20$	0.41	0.44	0.62	0.41

Table 1: Recall@100 reported for ablations for K and τ.

$$L(u_1, u_2, \{u_i\}_{i=1}^{N}) = -\log\left(\frac{\exp(\text{sim}(u_1, u_2)/\tau)}{\exp(\text{sim}(u_1, u_2)/\tau) + \sum_{i=1}^{K}\exp(\text{sim}(u_1, u_i)/\tau)}\right)$$

Where:

- $\text{sim}(u_i, u_j)$ denotes the similarity function between the embeddings u_i and u_j. We choose cosine similarity.
- τ is the temperature parameter that scales the similarity scores, affecting the sharpness of the softmax distribution. Lower values of τ increase the separation between positive and negative pairs, while higher values make the differences less pronounced. We find that the choice temperature has a significant impact on the results and adjust τ to $\tau = 0.07$.
- $\{u_i\}_{i=1}^{N}$ is a set of K negative embeddings used to contrast against the positive pair. We tune K to $K = 5$.

During training, we define the loss weight w for each instance as:

$$w(u_1, u_2) = \log(\text{CoClick\_Count}(u_1, u_2) + 1),$$

For each URL, we randomly sample positives from its neighbors in the CoClick graph (the set of URLs with which it has a CoClick count > 0). For each training tuple (URL1,URL2), we randomly sample the negative URL samples from all URLs.

Table 1 is a subset of our hyper-parameter ablations illustrating the importance of tuning the CF hyper-parameters well to maximize performance on our task. We find that the InfoNCE loss for our task is highly sensitive to both K and τ. Our CF training uses the stochastic gradient descent (SGD) optimizer, and we find that the learning rate scheduler and associated hyperparameters have a considerable impact on the quality of the learned embeddings (as evaluated by Recall@k). While optimizers such as rAdaGrad [16] and Adam [9] converge faster on our training data, we find that SGD trained on more iterations generalizes better on our test sets. We find the Step Decay learning rate scheduler [7] to be critical in obtaining strong results, and tune the hyperparameters extensively on a subset of the training data before kicking off the large-scale jobs (which take in the order of weeks to converge).

3.2.1 Sampling strategy during training: We have the option to randomly shuffle and use the complete set of interactions (clicks) in every epoch (interaction-based sampling) or instead to iterate over every URL1 in an epoch and uniformly randomly sample a URL2 from its CoClick neighbors (URL1-based sampling). Note that unlike interaction-based sampling, URL1-based sampling does not guarantee that all interactions will be used during training. However, in practice this concern is alleviated, since we train our CF implementation to convergence at around 10,000 epochs. Interaction-based sampling leads to head (popular) URL1s receiving significantly more gradient updates than tail URL1s, thus biasing the embeddings toward better performance on head URL1s. However, our CoClick path is already a robust candidate generator for URL1s that are popular and receive many clicks, and it is updated weekly. Therefore, our objective with CF is to develop a complementary candidate generator that can retrieve relevant recommendation candidates even for body or tail URL1s that receive few clicks and are not covered by the CoClick path. URL1-based sampling treats each URL1 equally per epoch, leading to learned embedding representations that perform significantly better on body and tail URL1s. Although initial versions of our CF implementation leveraged interaction-based sampling, we achieved a significant jump in performance with URL1-based sampling on our CF test set: a 3% absolute increase in Recall@100.

We mention these details to emphasize the point that a well-tuned traditional model can be extremely powerful in a real-world setting especially when training at scale with billions of interactions; we have experimented with several newer models that learn from user-item (or item-item) interactions but find it challenging to surpass both the performance and efficiency of this CF implementation.

3.2.2 *Collaborative Filtering at Scale*

To scale collaborative filtering (CF) to handle a billion-scale interaction matrix (CoClick counts), we developed a distributed implementation of our CF model in C++. Our approach leverages model parallelism by partitioning the training data and embeddings along a 2D grid. The embeddings are partitioned along the same row and column grid boundaries. Workers store multiple training data shards and only the embedding vectors representing documents in the local training data shards.

Stratified sampling allows distributed workers to train asynchronously on a local block without intermediate communications. After all workers train on a local shard, the updated embedding weights are exchanged and the next block begins [14]. The embedding communications are sparse such that only the sampled vectors are communicated in each stage to minimize bandwidth consumption. The stratified sampling and sparse communication allow the training to only store a single representation of the full model in memory; gradients are only allocated in a sparse form and never communicated. Finally, we use data parallelism within each block, where threads train on samples asynchronously without synchronization [11].

This distributed implementation enables us to train on a CoClick count matrix for 2 billion webpages, with dimensions of 2 billion by 2 billion, making it one of the largest collaborative filtering systems in the industry to date.

3.2.3 *Online Serving:*

We create a vector index FAISS [6] using the learned URL2 embeddings and inject the learned URL1 embeddings into a key-value store. At serving time, for a specific URL1, we retrieve its embedding from the key-value store and utilize Approximate Nearest Neighbor (ANN) search to obtain URL2 candidates.

3.3 SAT CF Recall Path

The collaborative filtering (CF) recall path discussed in the prior section involves training the model on Bing session CoClick data. We develop a new CF recall path leveraging user *satisfied* (SAT) clicks on our WebReco production system. We collect SAT click count data from our production logs, generating (URL1, URL2) pairs. With identical training settings, we learn the CF embeddings for 350 million URL1s and URL2s, training on a *satisfied* click count matrix of dimensions 350M by 350M URLs. We serve this recall path as a separate FAISS vector index, following the same online flow as described in Section 3.2.3. On deployment, this recall path led to a 2% relative increase in online CTR over our production baseline.

3.4 Two-tower Dense Retrieval Model

We fine-tune a two-tower 6-layer XLM-Roberta [5] based transformer encoding model. The two towers are fed URL1 and URL2 features, respectively. We share the URL1 tower and the associated full Bing embedding index with other teams (e.g., Web Search Retrieval). Consequently, during training, the URL1 tower is frozen and we optimize the parameters only for the URL2 tower. The input schema for each tower consists of the URL, Title, and Snippet concatenated with [SEP]. The title is generally the webpage HTML's primary title and the representative snippet is extracted from the webpage's text content with the help of text extraction models and finally a snippet ranker. We input this text sequence into XLMR, using the last layer's [CLS] token embedding followed by an MLP head to produce the final embedding vector. We employ the Info-NCE contrastive loss [10] for training, with a temperature parameter set to 0.02.

3.4.1 *Training Data:*

Training samples are sourced from our production system's click logs. Positive samples are generated as pairs of the form URL1-URL2 from an impression, where URL2 was clicked and had a dwell time exceeding a predefined threshold. While not a perfect heuristic, requiring the dwell time threshold helps improve the training data quality by ensuring that for that impression, the user's information need was met. For sampling negative URL2s, we include in-batch negatives and ANCE [17] hard negatives, which are randomly sampled from the top 100 to 300 webpages retrieved via Approximate Nearest Neighbor Search (ANNS).

3.4.2 *Building a SPANN index:*

We select URLs for our embedding index by ranking the top 500 million URLs based on aggregate click counts from 6 months of Bing search logs. The full Bing index is over 200 billion webpages, but given cost and latency considerations, for our feature, we limit the index to the top 500 million most engaged with URLs. We then use the URL2 tower to perform model inference and generate embeddings for these 500 million webpages and build a SPANN (Scalable and Parallel Approximate Nearest Neighbor) [3] index for fast ANN search.

3.4.3 *Online Serving:*

At serving time, for a given URL1 we fetch its embedding from the key-value store for the full Bing index. SPANN is designed specifically for billion-scale approximate nearest neighbor search, enabling us to search over our URL2 embedding index to retrieve the top 30 URL2s with minimal latency. To help with latency, we maintain a cache of the ANN search results with a cache expiration time of 6 hours to control the cache size.

3.5 LLM Powered Q' Recall

This recall path uses LLMs to generate diverse complementary website candidates relevant to key subtopics related to the URL1 trigger content. We designed and extensively tuned a Q' query generation prompt that leverages webpage content features namely the URL, title, and webpage body (processed and cleaned from the webpage HTML) to generate an array of (on average) 6 queries that cover a diverse set of aspects of the webpage content and complementary topics. We use Bing Web search as our retrieval engine. A high-throughput service was developed for efficient Bing scraping, followed by post-processing filters such as spam detection and deduplication. We obtain the top 10 result URLs for each generated query. For each URL1, we store an array of upto 30 URL2 candidates in a highly performant key-value store for online serving.

In order to scale Q' query generation inference to hundreds of millions of URLs, we turned towards fine-tuning Small Language Models (SLMs). We experimented with several LLMs and found Mistral-7B [1] the best suited for this task in terms of task performance and throughput. We first create a dataset of URLs and the associated GPT-4 generated Q' queries. We fine-tuned Mistral-7B, a 7.3 billion parameter model, with supervised fine-tuning (SFT) to match GPT-4's performance on the task, achieving a 60x increase in inference throughput.

With the fine-tuned Mistral-7B, we were able to substantially scale our Q' query generation and use our method to build an index supporting 600 million URL1s, resulting in significant improvements in recommendation quality (as evaluated with RecoDCG@5) as well as online CTR.

A/B Test	Online CTR	RecoDCG@5
Q' Recall (GPT-4)	+0.89%	+0.760
Q' Recall (Mistral-7B)	+1.71%	+0.762

Table 2: Online A/B results and RecoDCG@5 reported with the baseline system as control for all rows.

3.6 Fresh Recall Path

We define a query as fresh if it reflects the user's need for the latest information. We consider a URL a fresh webpage candidate if it is published (or updated) recently, meets a minimum pagerank [1] score threshold, and passes a set of quality filters such as spam and adult classifiers. The core challenge we address here is two-fold. First, thousands of fresh webpages are being published every second and the SPANN ANN embedding index that we describe in Section 3.4 is built for scale but cannot handle updates to its index at this rate. Furthermore, newly published fresh documents have not garnered sufficient impressions and user engagement and, as such, generally are not covered by our other recall paths. To address the fast update challenge, we build a new fresh recall path with a LightGBM query fresh intent classification model, a near real-time streaming pipeline to determine which URL1s we will trigger the fresh recall path for and leverage a streaming ANNS (Approximate Nearest Neighbor Search) index.

[1]https://mistral.ai/news/announcing-mistral-7b/

3.6.1 Query fresh intent classification model: Since we are constrained from increasing latency and GPU requirements for serving, we cannot deploy and serve a text model, such as a transformer, for this task. Instead, we developed a LightGBM model for the query fresh intent classification task: determining from the query whether meeting the user's information need requires the latest (fresh) documents. It is a LightGBM model trained with the BCE (Binary Cross Entropy) objective on around binary 30K Bing user queries with high-quality human judge labels. We do a lot of feature engineering to develop several strong query-level signals, such as:

(1) Average document age (the time since the most recent update to the page when crawled) over the top N retrieved webpages from the Bing search retrieval and ranking stack.
(2) The average of the top k ($k \in \{1, 3, 5, 10\}$) documents' ages when ranked by several Bing query-URL relevance and click prediction rankers (both transformer and LightGBM) models in our Bing ranking stack. These scores have already been computed and are available at this stage for the top N webpages. We find that features such as the difference between the average document age of the top k and the average document age of the bottom 3, computed for each of the Bing rankers, are also quite powerful.
(3) We compute features such as for the average Bing ranker scores of the top k documents ($k \in \{1, 3, 5, 10\}$) when ranked by document age.

3.6.2 Streaming Pipeline for collecting URL1s: Our near real-time streaming pipeline runs for each Bing search impression and helps collect a rolling set of URL1s on which our system's fresh recall path should trigger. It comprises of the following steps:

(1) The LightGBM model classifies the user query as fresh intent.
(2) For fresh intent queries, URL1 (top-ranked or clicked) undergoes spam, adult and popularity filters.
(3) We perform near real-time embedding inference using the URL1 tower from Section 3.4 and store URL1 slong with the embedding in a key-value store.

3.6.3 Streaming Similarity Search: The FreshDiskANN ANN Index In order to have an ANNS (Approximate Nearest Neighbor Search) algorithm that could accommodate real-time fresh document additions and fast removals based on expiry time without sacrificing ANN performance, we turned to FreshDiskANN [13]. We inject newly published or updated documents at the rate of thousands of webpages per second into this FreshDiskANN index. To control the size of the index, we set a 5-day expiration on the injected pages.

3.6.4 Online Serving flow: For any Bing impression, the online flow is the following:

(1) Check if the Web Reco trigger URL1 is present in the fresh URL1 key-value store populated by the streaming pipeline and fetch its URL1 embedding
(2) Conduct ANNS (Approximate Nearest Neighbor Search) using the FreshDiskANN index to obtain upto 30 URL2 candidates.

(3) Enforce quality checks on returned URL2 candidates such as a minimum URL1-URL2 embedding cosine similarity and a spam check, merge with candidates from the other recall paths, and send the candidates to the ranking stack.

3.6.5 Online Results Online A/B experiment results for introducing the fresh recall path show a +34.79% relative increase in PTR and +20.17% relative increase in CTR computed on the fresh segment (subset of all impressions with queries with a detected fresh intent), when compared to the production baseline.

4 Ranking

The Bing Web Recommendations ranking stage consists of, in order:

(1) Stage 1: LightGBM ranker with a set of inexpensive features, primarily Counting Features
(2) Stage 2.a: A multi-task transformer cross-encoder model with two heads that that are optimized for pairwise click prediction and recommendation quality tasks respectively
(3) Stage 2.b: LightGBM ranker with a more comprehensive set of features, including the cross-encoder model scores
(4) Stage 2.c: Linear Combination between the Stage 2 LightGBM ranker score and the recommendation quality head score

In the following section, we describe counting features, the most important feature set used in our ranking stack.

4.1 Counting Features

4.1.1 Definitions:

Quickback: A quickback refers to a user click where their information need was not satisfied, leading them to return to the Bing search page. Based on user analysis, clicks with a dwell time of less than 20 seconds are considered quickbacks.

Counting Features: Counting features refer to click, quickback, and impression counts aggregated at the levels of URL1-URL2, URL1, and URL2.

4.1.2 URL1-URL2 Aggregation: We process our production logs to generate counting features over three rolling time windows: the past 7 days, 28 days, and 168 days. As described in Section 1, our product has two surfaces: the inline scenario and the clickback scenario. Since user behaviors differ between the two scenarios, we calculate both a scenario-specific set of counting features and aggregate counting features across both scenarios. We further track position-specific click and impression counts, generating increasingly granular features such as `QuickbackCount_28d_clickback` or `ClickCount_7d_position_0_inline`.

4.1.3 Daily Counting Features: We also generate daily counting features for each of the previous 6 days, such as `ClickCount_1d_d1` (1-day counting feature for the previous day) and `ClickCount_1d_d2` (1-day counting feature for two days ago).

4.1.4 URL1 Aggregation: We further aggregate the URL1-URL2 counting features by grouping them by URL1 to generate features such as `ClickCount_7d_URL1`, which represents "the number of times in the past 7 days that an Explore Further recommendation tied to URL1 received a click".

4.1.5 URL2 Aggregation: Similarly, we aggregate the URL1-URL2 counting features by grouping them by URL2 to generate features such as `ClickCount_7d_URL2`, which represents "the total number of times in the past 7 days that URL2 was recommended and received a click, across all URL1s".

4.1.6 CoClick counting features: We also inject the URL1-URL2 CoClick count described in Section 3.1, as it proved to be a valuable feature to our LightGBM rankers.

4.1.7 Online Serving: Our counting features are updated on a daily cadence, and we illustrate the pipeline in Figure 5. For online serving, all counting features (aggregated on URL1-URL2, URL1, and URL2) are injected into performant key-value stores based on their aggregation key.

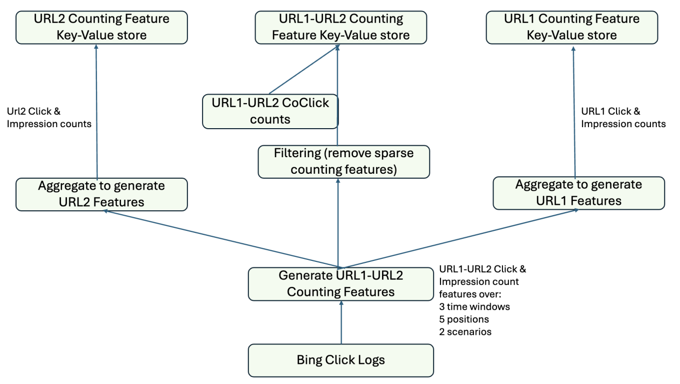

URL2 Counting Feature Key-Value store
URL1-URL2 Counting Feature Key-Value store
URL1 Counting Feature Key-Value store
Url2 Click & Impression counts
URL1-URL2 CoClick counts
URL1 Click & Impression counts
Filtering (remove sparse counting features)
Aggregate to generate URL2 Features
Aggregate to generate URL1 Features
Generate URL1-URL2 Counting Features
URL1-URL2 Click & Impression count features over: 3 time windows 5 positions 2 scenarios
Bing Click Logs

Figure 5: Counting Features Daily Pipeline

4.2 LightGBM Rankers:

We have two LightGBM [8] rankers, Stage 1 and Stage 2. As shown in the system diagram in Figure 3, the Stage 1 LightGBM ranker provides the top 30 documents after scoring the few hundred candidates returned by the candidate generation stage and was one of the first rankers we introduced to our system. Since our feature appears on the first page of Bing search results, there are hard latency requirements. As such, since the Stage 1 LightGBM ranker scores several hundred webpages, it consumes a smaller set of features that are easily available at this stage: URL1-URL2 aggregated counting features and embedding cosine similarity from the two-tower model. Since the Stage 2 LightGBM ranker scores only 30 candidates, it leverages a much larger feature set: including the cross-encoder model scores, URL2 characteristics (depth, length), text attributes (title and snippet length, highlight count), URL1-URL2 counting features and their interaction features (for e.g. 7d CTR, 28d CTR, 168d CTR). When first introduced, Stage 2 LightGBM deployment achieved a gain of + 1.25% in CTR. This underscores the value of implementing a lightweight tree-based Learning to Rank (LTR) model that integrates both counting featues that describe historical user interaction behavior and RecoLM scores that are computed using the text signals. In fact, the RecoLM pairwise click prediction head score is the most important feature by both split count and gain for the Stage 2 LightGBM ranker.

The LightGBM rankers are trained on several weeks of historical logs from our production system. The training labels are obtained by applying a piecewise utility function that transforms the logged dwell time. The utility function is designed to penalize extreme quick backs, defined as clicks with a dwell time of 0-5 seconds, which are indicative of user dissatisfaction with the clicked URL2. Therefore, non-clicked documents are assigned a higher label than those associated with quick backs. Beyond the initial 5 seconds, the function increases linearly until reaching the 30-second mark, at which point the click is considered a *satisfied* click. At this threshold, the utility value is further enhanced by adding a constant boost. The utility function of the piecewise function is illustrated by the plot in Figure 6.

4.2.1 LightGBM ranker improvements: This section describes a set of ranker improvements we sequentially added to our production LightGBM rankers, achieve online click metric gains.

- Since we train on non-aggregated raw logs, we add instance weights based on traffic, giving a low weight to highly frequent (head) URL1s and a large weight to less frequent (tail) URL1s.
- We add URL2 aggregated counting features and the derived CTRs as features to both LightGBM rankers. We observe that the URL2 aggregated counting features are considerably denser than the URL1-URL2 aggregated counting features and help significantly with generalization.
- Initially, our deployed LightGBM rankers used label regression (MSE loss) as the learning objective. Shifting to a pairwise objective, LambdaRank, led to significant online gains. LambdaRank is a pairwise objective: for each impression, for the URL1, we consider a pair of the shown candidate URL2s. Specifically, in LambdaRank, the pairwise objective for a URL1 and candidates URL2$_i$ and URL2$_j$ is given as:

$$\log\left(1+\exp^{-\sigma(s_i-s_j)}\right)\left|\Delta NDCG_{ij}\right|$$

  Where:
  - s_i and s_j are the model scores for URL2$_i$ and URL2$_j$, respectively.
  - $\Delta NDCG_{ij}$ represents the NDCG difference that results from swapping the ranking of the pair. The relevance labels used for calculating the $\Delta NDCG_{ij}$ are the utility labels for the URL2s computed as a function of their dwell times.

We provide ablations for these techniques for the Stage 2 LightGBM ranker in Table 3. We evaluate the techniques with the ClickNDCG inner loop metric described in Section 2.2. The final productionized Stage 2 LightGBM Ranker (last row of Table 3) showed a 0.83% relative CTR gain on a 1 week online A/B experiment.

Technique	ClickNDCG@3 Delta
Traffic Instance Weights	+0.02
Previous row + URL2 aggregated counting features	+0.13
Previous row + update objective to LambdaRank	**+0.18**

Table 3: Ablation results for techniques used in the Stage 2 LightGBM ranker on the offline test set

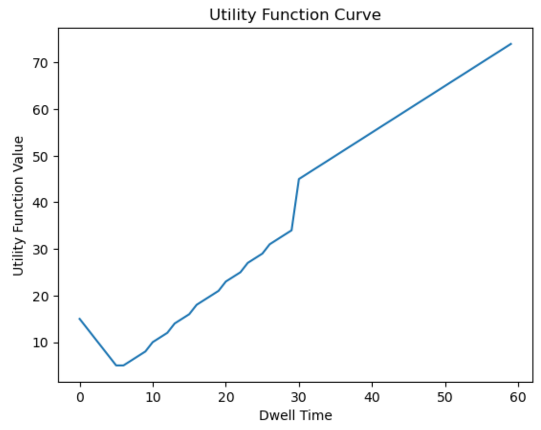


Figure 6: Utility label for LambdaRank: piecewise function plot

4.3 RecoLM: A Multiobjective Cross-Encoder Model

MT RecoLM is a multi-task MiniLM-based cross-encoder model designed for ranking URL1-URL2 pairs. We cover several aspects of the model development in this section: the teacher model pre-training, teacher fine-tuning, and knowledge distillation.

Teacher Pretraining and Domain Adaptation: We use TuLRv3 [4] as our base model, pretraining it with billions of Bing search logs. We apply Contextual Masked Language Modeling along with pairwise objectives for dwell time and URL position prediction. The input during pre-training is <Query, URL, Title, Snippet>, adapting the model for web search applications across Bing teams.

Click Teacher: We build on the pre-trained model by adding a click prediction head on the [CLS] token. We collect several months of production logs and derive the unique (URL1, URL2_clicked and URL2_not_clicked) triplets, amounting to over 1 billion training triplets. The input to the model during finetuning is of the form <URL1, Title1, Snippet1, URL2, Title2, Snippet2>. Training data consists of triplets of clicked and non-clicked URLs, and we fine-tune on approximately 1 billion triplets. Table 5 summarizes the evolution of our click teacher model. For first iteration of the click teacher, we bucketized the 60-day CTR into 5 classes and trained the model with Cross-Entropy loss, a pointwise objective. On changing the objective to softmax pairwise loss, we observed a significant gain of +1.45 points on our offline ClickNDCG@3 metric as shown in Table 5.

$$\text{Pairwise Softmax Loss} = \log\left(\frac{e^{s_{\text{Url1\_Url2}^+}}}{e^{s_{\text{Url1\_Url2}^+}}+e^{s_{\text{Url1\_Url2}^-}}}\right) - \log\left(\frac{e^{s_{\text{Url1\_Url2}^-}}}{e^{s_{\text{Url1\_Url2}^+}}+e^{s_{\text{Url1\_Url2}^-}}}\right)$$

Where:

- $s_{\text{Url1\_Url2}^+}$ is the model score for the positive (clicked) URL2.
- $s_{\text{Url1\_Url2}^-}$ is the model score for the negative (not-clicked) URL2.

During fine-tuning with pairwise loss, we further introduced loss importance weights for each (URL1, URL2_clicked, URL2_not_clicked) training instance. The importance weight for each instance is calculated as the logarithm of the frequency f of the triplet in the collected training logs. Using a logarithmic function rather than a linear weight helps mitigate the impact of highly frequent triplets corresponding to extremely popular head URL1s, thus preventing them from disproportionately influencing training. Introducing importance weighting yields another +0.5 gain in ClickNDCG@3.

Recommendation Quality Teacher: The quality teacher, built on top of the same pretrained model, includes a classification head with 5 logits for the recommendation quality classes 0 to 4. We obtain 80 million URL1-URL2 recommendation quality scores (0 to 4) with GPT-4 Turbo, using the pointwise prompt discussed in Section 2.1. For fine-tuning, we use cross-entropy loss, rounding the quality scores to the nearest integer.

4.4 MT RecoLM: Knowledge Distillation

The RecoLM student model was originally a single objective model with a single logit click prediction head. It evolved to become a multi-objective model with the addition of a 5 logit recommendation quality classification head. To meet latency requirements for online serving, we distill a 6-layer MiniLM-based cross-encoder, MT RecoLM, from a significantly larger teacher model. We use a dataset of 1 billion (URL1, URL2) pairs with click and quality teacher labels for distillation. The student model features two heads: one for pairwise click prediction and one for quality classification, both trained with MSE loss and we tune the loss weights for the best joint performance. We also show through the experiment summarized in Table 4, that distillation outperforms direct fine-tuning of the previous version of RecoLM (by adding a quality head to it) with SCE loss by over 8 points in RecoDCG@5, demonstrating its effectiveness for multi-objective ranking models.

Note: In Table 4, RecoDCG@5 is reported by ranking webpages by the quality head scores of the MT RecoLM student model. Both approaches are compared against the baseline single objective student model (optimized solely for clicks) performance.

Model	**RecoDCG@5**
Teacher Distillation (MSE Loss)	+9.76
Direct Fine-tuning (SCE Loss)	+1.44

Table 4: MT RecoLM student model: Teacher distillation versus fine-tuning

4.5 Linear Combination

The LightGBM ranker score and the multi-task student model's quality head scores are first Z-score normalized using the mean and standard deviation calculated on a fixed validation set that represents production traffic. We deploy a linear combination between the stage 2 LightGBM ranker score and the quality head score as:

$$(1-\lambda)\cdot\frac{\text{lgbm\_score}-\mu_{\text{lgbm\_score}}}{\sigma_{\text{lgbm\_score}}}+\lambda\cdot\frac{\text{cls\_score}-\mu_{\text{cls\_score}}}{\sigma_{\text{cls\_score}}}$$

Where:

- λ: quality weight
- cls_score: recommendation quality head score
- lgbm_score: Stage 2 LightGBM ranker score

The linear combination provides a controllable and interpretable mechanism to balance the trade-off between quality improvement (RecoDCG) and online click metrics, using the weight λ.

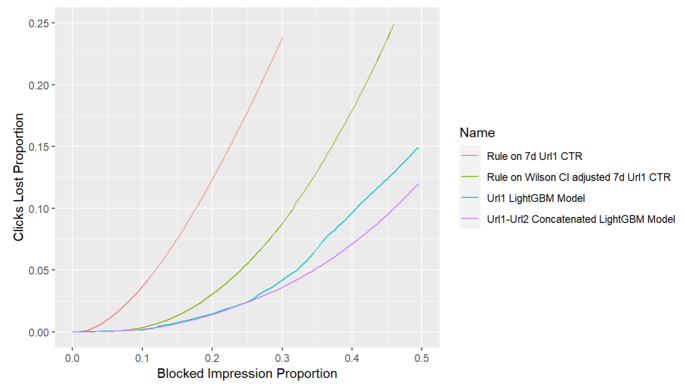


Figure 7: Evolution of the Triggering Model: Clicks Blocked Proportion vs. Impressions Blocked Proportion

5 Triggering Model

Historically, for all URL1s and impressions we've always shown 5 recommendations in the inline scenario. The motivation behind the triggering model is to improve the user experience on Bing in situations where showing all 5 WebReco recommendations may be ineffective. This includes navigational queries, where user intent is highly specific, or situations where the quality of our recommendations falls short and do not engage users. For example, with queries such as 'definition of irrelevant' or 'YouTube', the primary search result typically meets user needs, which makes showing 5 WebReco recommendations an inefficient use of space on the Bing search result page. The trigger model aims to boost online engagement metrics (PTR and CTR) and enhance the overall Bing experience by dynamically adjusting the number of recommendations shown—selecting from $\{3, 4, 5, 6\}$—based on the specific URL1. The simple underlying intuition is to decrease the recommendation count if the historical CTR when showing recommendations for a URL1 is low and to increase the count if the historical URL1 CTR is high. We impose the constraint of keeping the average number of recommendations shown by our production system constant. This is because we do not wish to over-trigger recommendations to achieve PTR and CTR gains as user experience critical and there are other features and answer blocks on the Bing results page that provide value. Instead, we want to more optimally allocate recommendation counts to URL1s who would most benefit from it, by reducing the recommendation counts allocated to URL1s who have historically had low engagement.

5.0.1 Offline Threshold Tuning: We developed an offline threshold-tuning pipeline using one week of production logs as the evaluation

Version	Type	Training Data	Label Type	Objective	ClickNDCG@3
V1	Click	URL1-URL2 Counting Features	Bucketized 60 day CTR	Cross-Entropy Loss	61.49
V2	Click	Aggregated Production logs	Dwell times	Pairwise Softmax Loss	62.94
V3	Click	Aggregated Production logs	Dwell times	Pairwise Softmax Loss with Importance Weighting	63.44

Table 5: Evolution of the RecoLM Click Teacher model

set. Given a target recommendation count distribution and a scoring model, the pipeline determines the score threshold buckets for each recommendation count $\in \{3, 4, 5, 6\}$ by creating a cumulative distribution of the total URL1 impressions from the 1 week evaluation set. We determine precise thresholds on the score, with the aim of achieving the target recommendation distribution for the recommendation counts $\in \{3, 4, 5, 6\}$. We apply these thresholds on top of the rule to go from the scoring model's score to a predicted recommendation count for each impression's URL1.

5.0.2 Rules-based Triggering Models: The first version of the triggering model was rule-based and given a target recommendation count distribution, we use the aforementioned threshold-tuning pipeline to find the required thresholds on the 7-day rolling URL1 CTR. Naturally, however, there is greater confidence in the 7-day CTR for a URL1 with a high impression count compared to one with a low impression count.

To account for this, we compute the Wilson Confidence Interval upper bound adjusted URL1 7-day CTR shown in Equation 5.0.2, at the significance level 95%. This provides a confidence interval about the CTR and is scaled based on the 7-day URL1 impression count. The upper and lower Wilson bounds [2] are detailed below.

$$\text{Upper Bound} = \frac{\hat{p} + \frac{z^2_{1-\alpha/2}}{2n} + z_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2_{1-\alpha/2}}{4n^2}}}{1 + \frac{z^2_{1-\alpha/2}}{n}}$$

$$\text{Lower Bound} = \frac{\hat{p} + \frac{z^2_{\alpha/2}}{2n} + z_{\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2_{\alpha/2}}{4n^2}}}{1 + \frac{z^2_{\alpha/2}}{n}}.$$

Where:

- $\hat{p}$ = URL1 CTR
- n = URL1 Impression Count
- z = Test statistic for $\hat{p}$
- α = Significance level, 95%

5.0.3 Comparing Triggering Model Strategies: We collect 1-day of production logs collected after the training data window. To visualize and compare the effectiveness of different triggering rules and models in a 2-D plot, we assume a binary decision based on a threshold t on whether to block URL1 from triggering. We then plot the proportion of blocked clicks against the proportion of blocked impressions, varying the blocking threshold t. A more effective model minimizes impressions while retaining the highest possible number of clicks, resulting in a lower curve. In Figure 7, the rule that uses the Wilson upper bound adjusted 7-day CTR performs significantly better than the baseline rule on the raw 7-day URL1 CTR. This is corroborated by strong CTR and PTR gains in an online A/B experiment comparing these two rules.

5.0.4 Target Recommendation Count Distribution: We want the triggering model to respect a target recommendation count distribution such that the average number of recommendations shown to users remains the same compared to the production baseline (control), about 4.9 recommendations per impression. We eventually choose the following distribution: show 3 recommendations for 21% of URL1s, 4 recommendations for 20%, 5 recommendations for 20%, and 6 recommendations for 39% of URL1s. These percentages are weighted by the URL1 7-day impression count.

5.0.5 LightGBM-based Triggering Models: The next iteration of the triggering model, referred to as the "URL1 Model" in Figure 7, advanced beyond the rule-based approach by incorporating a LightGBM model. This model leverages the 7-day URL1 aggregated counting features, along with Wilson upper and lower bound adjusted URL1 CTRs for the same window, to predict the future 1-day Wilson upper bound adjusted URL1 CTR. The model is trained with a regression objective using Mean Squared Error. The final and our best-performing model ("URL1-URL2 Concatenated" model) uses an even richer feature set: it further includes URL1-URL2 counting features and the predicted Stage 2 LightGBM scores for each of the top 5 ranked URL2 candidates for the URL1. This "URL1-URL2 Concatenated" model demonstrates the best performance in Figure 7, achieving the best tradeoff between impressions blocked and clicks lost. This is our final model, demonstrating a relative gain of 2.11% CTR / 1.93 % PTR in online A/B experiments.

The engineering challenge to ship the "URL1-URL2 Concatenated" triggering model to production is that the thresholds mentioned must be tuned at a regular cadence, since the triggering model when active will affect the counting features' distribution.

6 Conclusion

The paper provides an in-depth examination of WebReco, from candidate generation to ranking. Our work details our end-to-end production system through an inside look into the real-world challenges and solutions when building a web-scale recommendation system. For candidate generation, we share our insights into the development of our billion-scale CoClick, collaborative filtering (CF), dense retrieval, fresh and LLM-powered Q' recall paths. The paper details how the ranking models evolved and how the techniques we introduced improved the ranking system. We also motivate the triggering model and discuss its evolution. By sharing our system architecture and the design choices made along the way, we aim to offer actionable insights and strategies for practitioners developing scalable recommendation systems that align with their product goals.

References

[1] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine". In: *Computer Networks and ISDN Systems* 30.1-7 (1998). Proceedings of the Seventh International World Wide Web Conference, pp. 107–117. ISSN: 0169-7552. DOI: 10.1016/S0169-7552(98)00110-X. URL: https://www.sciencedirect.com/science/article/pii/S016975529800110X.

[2] Lawrence D. Brown, T. Tony Cai, and Anirban Dasgupta. "Interval Estimation for a Binomial Proportion". In: *Statistical Science* 16 (2001), pp. 101–133. URL: https://api.semanticscholar.org/CorpusID:7039587.

[3] Qi Chen et al. *SPANN: Highly-efficient Billion-scale Approximate Nearest Neighbor Search*. 2021. arXiv: 2111.08566 [cs.DB]. URL: https://arxiv.org/abs/2111.08566.

[4] Zewen Chi et al. *InfoXLM: An Information-Theoretic Framework for Cross-Lingual Language Model Pre-Training*. 2021. arXiv: 2007.07834 [cs.CL].

[5] Alexis Conneau et al. *Unsupervised Cross-lingual Representation Learning at Scale*. 2020. arXiv: 1911.02116 [cs.CL].

[6] Matthijs Douze et al. *The Faiss library*. 2024. arXiv: 2401.08281 [cs.LG].

[7] Rong Ge et al. *The Step Decay Schedule: A Near Optimal, Geometrically Decaying Learning Rate Procedure For Least Squares*. 2019. arXiv: 1904.12838 [cs.LG]. URL: https://arxiv.org/abs/1904.12838.

[8] Guolin Ke et al. "LightGBM: a highly efficient gradient boosting decision tree". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. Long Beach, California, USA: Curran Associates Inc., 2017, pp. 3149–3157. ISBN: 9781510860964.

[9] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[10] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. *Representation Learning with Contrastive Predictive Coding*. 2019. arXiv: 1807.03748 [cs.LG].

[11] Benjamin Recht et al. "Hogwild!: A lock-free approach to parallelizing stochastic gradient descent". In: *Advances in neural information processing systems* 24 (2011).

[12] Steffen Rendle et al. *BPR: Bayesian Personalized Ranking from Implicit Feedback*. 2012. arXiv: 1205.2618 [cs.IR]. URL: https://arxiv.org/abs/1205.2618.

[13] Aditi Singh et al. *FreshDiskANN: A Fast and Accurate Graph-Based ANN Index for Streaming Similarity Search*. 2021. arXiv: 2105.09613 [cs.IR]. URL: https://arxiv.org/abs/2105.09613.

[14] Shaden Smith, Jongsoo Park, and George Karypis. "An exploration of optimization algorithms for high performance tensor completion". In: *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE. 2016, pp. 359–371.

[15] Paul Thomas et al. *Large language models can accurately predict searcher preferences*. 2024. arXiv: 2309.10621 [cs.IR]. URL: https://arxiv.org/abs/2309.10621.

[16] Minhui Xie et al. "Kraken: Memory-Efficient Continual Learning for Large-Scale Real-Time Recommendations". In: *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2020, pp. 1–17. DOI: 10.1109/SC41405.2020.00025.

[17] Lee Xiong et al. *Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval*. 2020. arXiv: 2007.00808 [cs.IR]. URL: https://arxiv.org/abs/2007.00808.

A Appendix

GPT-4 Generated Queries
1- Florida Lottery education and community initiatives
2-Florida Lottery promotions and events
3- Florida Lottery winners stories and testimonials
4- How to play and win the Florida Lottery games
Mistral-7B Generated Queries
1- Florida Lottery games and prizes
2- Florida Lottery promotions and news
3- Florida Lottery winners and stories
4- How to play and win the Florida Lottery

Table 6: GPT-4 and Mistral-7B generated queries for Florida's lottery website (www.flalottery.com)

Received 8 August 2024; revised 8 August 2024; accepted 8 August 2024