International Journal of Advanced Research in Information and Communication Engineering (IJARCSA) ISSN: 2321-8940 Vol.5, Issue No 2, 2025

Serenade - Low-Latency Session-Based Recommendation in e-Commerce at Scale

V.Chiranjeevi [1], P.Ashwini [2], S.Nagamani [3]

[1]Assistant Professor, Swarna Bharathi Institute of Science & Technology, India, E-mail: chiru508@gmail.com.[2]Assistant Professor, Swarna Bharathi Institute of Science & Technology, India, E-mail: ashwini.podila@gmail.com.
[3]Assistant Professor, Swarna Bharathi Institute of Science & Technology, India, E-mail: nagamanikunchipudi@gmail.com.

Abstract:

Session-based recommendation forecasts the subsequent item a user will engage with, based on a sequence of her prior interactions with previous things. This machine learning topic addresses a fundamental scenario in e-commerce systems, which seek to propose appealing goods for purchase to customers navigating the site. Session-based recommenders are challenging to scale because of their exponentially vast input space of possible sessions. This hinders offline precomputation of recommendations and necessitates the maintenance of state throughout the online calculation of subsequent item recommendations. We present VMIS-kNN, an adaption of a cutting-edge nearest neighbor methodology for session-based recommendation, which utilizes a preconstructed index to provide next-item suggestions with minimal latency in contexts involving hundreds of millions of clicks to sift through. Utilizing this methodology, we develop and deploy the scalable session-based recommender system Serenade, now operational at bol.com, a prominent European e-commerce site. We assess the predictive efficacy of VMIS-kNN and demonstrate that Serenade can process one thousand recommendation requests per second, achieving a 90th percentile latency of under seven milliseconds in contexts including millions of objects for suggestion. Additionally, we provide findings from a three-week online A/B test, accommodating up to 600 requests per second for 6.5 million unique goods across over 45 million user sessions on our e-commerce platform. We provide the first empirical evidence that the enhanced predictive efficacy of closest neighbor methods for session-based recommendations in offline assessments corresponds to greater performance in a real-world e-commerce context.

INTRODUCTION:

Session-based recommendation addresses a fundamental situation in e-commerce and online navigation. Based on a series of interactions between a visitor and a selection of objects, we want to propose the subsequent item(s) of interest for the user to engage with [27, 29, 30, 37]. This machine learning issue is vital for e-commerce platforms [24]. Obstacles in expanding session-based recommendation systems. Scaling session-based recommender systems presents a formidable challenge due to the exponentially large input space (sequences of item interactions), quantified as |I|^n for all potential sessions of length n from a set of items I. This complexity renders precomputing recommendations offline and retrieving them from a data store impractical. This sharply contrasts with traditional collaborative-filtering suggestions, which are rather static due to their dependence on long-term user behavior. Conversely, session-based recommenders must preserve state to respond to online changes in dynamic user sessions and provide next item recommendations with minimal latency in real-time. Recent studies demonstrate that closest neighbor algorithms provide superior performance for session-based recommendations, surpassing intricate neural network-based approaches in offline assessments [24, 30]. It is ambiguous if this enhanced offline performance correlates with heightened user engagement in practical recommender systems. Moreover, it remains ambiguous whether academic closest neighbor methodologies can be scaled for industrial applications, which need the efficient examination of hundreds of millions of historical

clicks while complying with stringent service-level agreements on response latency. The scalability challenge is exacerbated by the use of session similarity functions that do not form a metric space (e.g., owing to asymmetry), rendering conventional approximation closest neighbor search methods ineffective. VMIS-kNN. To address the scalability difficulty, we introduce Vector-Multiplication-Indexed-Session-kNN (VMIS-kNN) in Section 3, an adaptation of the advanced session-based recommendation algorithm VS-kNN [30]. VMIS-kNN utilizes a preconstructed index to provide next-item suggestions within milliseconds for contexts involving hundreds of millions of clicks in previous sessions for analysis. Our methodology may be seen as the simultaneous execution of a join between evolving and historical sessions on corresponding items, together with two aggregations to calcula te the similarities. In this collaborative execution, we reduce intermediate results, manage memory consumption, and limit the search space by early termination. Consequently, VMIS-kNN significantly surpasses VS-kNN regarding latency and scalability (Section 5.2.1), while maintaining superior prediction quality compared to neural network-based methods (Section 5.1.1).Evaluation conducted both offline and online. We do a comprehensive assessment to verify the prediction efficacy and minimal latency of VMIS-kNN in Section 5.1. In Section 5.2, we report results from a load test of the Serenade system, which handled over 1,000 requests per second, as well as the findings from a three-week online A/B test conducted on the live e-commerce platform. Our system is accessible under an open license at https://github.com/bolcom/serenade. In conclusion, we provide the subsequent contributions. We provide VMIS-kNN, an index-based form of a cutting-edge nearest neighbor algorithm for session-based recommendations, capable of scaling to scenarios involving hundreds of millions of clicks for search (Section 3). We examine the design choices and implementation specifics of our production recommender system, Serenade, which utilizes stateful session-based recommendation with VMIS-kNN and is capable of processing over 1,000 requests per second, achieving a response latency of under seven milliseconds at the 90th percentile (Section 4). We present the inaugural empirical evidence demonstrating that the enhanced predictive efficacy of VMIS-kNN/VS-kNN, as established through offline evaluations, is reflected in superior performance within a real-world e-commerce context; our findings indicate that Serenade significantly elevates a business-specific engagement metric by several percentage points in comparison to our legacy system (Section 5.2.3).

BACKGROUND:

We provide session-based recommendation and the Vector-Session-kNN methodology. The objective of session-based recommendation is to precisely forecast the subsequent item with which the user will engage at time t + 1, based on an ongoing session (a series of interactions with a collection of items I) at time t. Vector-Session-k-Nearest Neighbors. Vector-Session kNN (VS-kNN) [30] is a cutting-edge closest neighbor method for session-based recommendation, surpassing existing deep learning techniques for this purpose. In VS-kNN, we own a collection of historical sessions. the developing session (Lines 5 & 6). Subsequently, we calculate the k nearest sessions Ns from Hs based on the similarity π (ω (s(t)))Th (Line 7), which utilizes an element-wise decay function π on the entries representing the insertion order in the dynamic session. All items present in the adjacent sessions are ultimately evaluated (Lines 8 & 9) by aggregating their similarities (the previously calculated decayed dot product) weighted by a non-linear function λ applied to the position max(ω (s(t)) $\odot$ n) of the most recent common item between the evolving session s(t) and the neighboring session n. The contribution of session similarity is further adjusted by a factor of one divided by the session length, and by a factor of one plus the logarithm of the inverse document frequency |H | of the item, where hi represents the number of historical sessions that include item i, a standard method in information retrieval to diminish the prominence of frequently occurring items. The indicator function 1n(i) equals one if item i is present in historical session n, and zero otherwise. Illustrative example. We provide a toy example for the calculation of session similarity and match weighting performed by VS-kNN. Consider a developing session s(t) = [0 1 1 0 1], which denotes interactions with the three items [1, 2, 4], and a historical session h = [0 0 1 0 1], indicating

interactions with the items [2, 4]. The function ω provides the temporal insertion sequence for the evolving session, for instance, ω (s(t)) = [0 1 2 0 3] of the elements in s(t), commencing with the first item (item 1 with insertion time 1) and concluding with the latest item (item 4 with insertion time 3). The insertion order is used to provide weights to matches between items in the developing session and the historical session, with the weights governed by the decay function π, a hyperparameter of VS-kNN. A prevalent option for π is to calculate the insertion time relative to the session duration, for instance, π (ω (s(t))i) = ω (s(t))i / ||s(t)||1. The similarity is ultimately ascertained by calculating the decayed dot product π (ω (s(t)))Th between the evolving session s(t) and the historical session h, represented as the sum of the decayed weights for the overlapping elements of the sessions (the common items), for instance, Three three three Sessions H ∈ {0, 1} |I | are represented as binary vectors inside the item space, whereas a developing user session s(t) ∈ {0, 1} |I | at time t is defined, along with a function ω(s) that substitutes the non-zero elements of s with integers indicating the insertion order of the items in s(t). Algorithm 1 delineates the methodology by which VS-kNN generates suggestions for an evolving session s(t). A recency-based sample Hs of size m is first extracted from all past sessions Hs that include at least one common item. Upon completing the session similarity calculation, VS-kNN derives item scores from the computed similarities (Lines 8 & 9). The score for an item is the weighted sum of similarities with s(t) from the k nearest past sessions n ∈ Ns in which the item appears. The weights for this summation are determined by the matching function λ, which is applied to the insertion time max(ω (s(t)) ⊙ n) of the most recent shared item between s(t) and n. The standard configuration The selection of λ in VS-kNN is defined as 1 − (0.1 · (max(ω (s(t)) ⊙ n))) for insertion times under 10, and zero in all other cases. In our toy example, the contribution of the matching function for h is as follows: λ (max(ω (s(t)) ⊙ h)) = λ (max([0 1 2 0 3] ⊙ [0 0 1 0 1])) = λ (max([0 0 2 0 3])) = λ (3) = 0.7.

VECTOR-MULTIPLICATION INDEXED SESSION-KNN (VMIS-KNN)

In the following, we present our scalable, index-based adaption of VS-kNN, which we call *Vector-Multiplication-Indexed-Session-kNN* (VMI kNN).VMIS-kNN operates on an index structure (M, t), which we build from a large dataset of historical sessions. We create a hash index.M from an item i to an array mi of the m most recent historical sessions in which the item occurs. Note that m is a hyperparameter of VMIS-kNN, which denotes the size of the recency-based sample from which session similarity candidates are taken. Each array mi of session identifiers for an item i is stored in descending timestamp order of the sessions (i.e., the most recent historical session h that contained the item i is the first entry in the vector mi). The key benefit of this data structure is to allow us amortised constant-time access to the m most recent sessions containing an item.

Algorithm 1 Vector-Session-kNN

1: **function** vs-knn(s(□), H, □ □ □ □)
2: *Input:* Evolving session s(□), set of historical sessions H, decay function □,
3: match weight function □ sample size □ number of neighbors □ 4:
Output: Scored list of recommended next items d.
5: H□ ← historical sessions that share at least one item with s
6: H□ ← recency-based sample of size □ from H□
7: N□ ← □closest sessions h ∈ H□ according to similarity □(□(s(□))ᵀh
8: **for** each item □ occuring in the sessions N□ **do**
9: □ ← 1□(□) · 1/(…) · □(max(□(s(□)) ⊙ n)) ·
n∈N□ □(□(s(□))ᵀn · (1 + log …)
return item scores d

Index-based session similarity computation:

Computation of session similarity based on indexing. The core of VMIS-kNN is in the efficient calculation of the neighbor sessions Ns for a developing session s(t) using our previously established index structure (M, t) inside the function neighbor_sessions_from_index in Line 8. Initially, we establish a collection of temporary hashmaps and heaps (Line 11) that function as buffers for intermediate results during the computation. Subsequently, VMIS-kNN initiates the item intersection loop, which traverses the items in a developing session s(t) in reverse order (Line 12). Our methodology analyzes a developing session s(t) in reverse insertion order, ensuring that the most recent and hence most significant elements are prioritized for

visitation. Subsequently, we include the item identification i into the temporary hashset d, so allowing for the exclusion of duplicate items inside the growing session (Lines 13-14). Subsequently, we progressing session (Line 16).

Algorithm 2 Vector-Multiplication-Indexed-Session-kNN.

```
1: function VMIS-knn(s', (M, t), □ □ □ □)
2:   Input: Evolving session s', session similarity index (M, t), decay function □,
3:     sample size □ match weight function □ number of neighbors □ 4:
     Output: Scored list of recommended next items d.
5:   (N_s, r) ← neighbor_sessions_from_index(s', (M, t), □ □ □)
6:   for each item □ occuring in the sessions N_s do
7:     □_i ← Σ_{j∈ 1:(□)} · □(max(□(s')) ⊙ n)) · □_i · log |L|/k_□
     return item scores d
8: function nEIgHBoR_SESSIonS_FRom_InDEx(s', (M, t), □ □ □)
9:   initialize hashmap r for temporary similarity scores, min-heap b_□ of cap-
10:  acity □for the most recent similar historical sessions, hashset d for already
11:  processed items, max-heap N_□ of capacity □for closest sessions
12:  for item □ ∈ s' in reverse insertion order do ◃ Item intersection loop
13:    if □ ∉ d then
14:      insert □ into d
15:      m_□ ← most recent sessions for item □ from inverted index M
16:      □_□ ← decay weight □(□(s'))_□ of item □ in session s'
17:      for session □ ∈ m_□ do
18:        if □ ∈ keys(r) then □_□ ← □_□ + □_□
19:        else
20:          □_□ ← timestamp of session □ fetched from index t
21:          if |r| < □ then
22:            □_□ ← □_□
23:            insert (□, □_□) into r
24:            insert (□, □_□) into b_□
25:          else
26:            (□, □_□) ← current heap root of b_□
27:            if □_□ > □_□ then
28:              □_□ ← □_□
29:              remove (□, □_□) from r
30:              insert (□, □_□) into r
31:              update heap root of b_□ with (□, □_□)
32:            else break
33:  for (□, □_□) ∈ r do ◃ Top-k similarity loop
34:    if |N_□| < □ then insert (□, □_□) into N35:
       else
36:      (□, □) ← current heap root of N_□
37:      if □_□ > □_□ then update heap root of N_□ with (□, □_□)
38:      else if □_□ = □_□ and □_□ > □_□ then update heap root of N_□ with (□, □_□)
39:  return N_□
```

We enter the session identification j and session timestamp t j as a (key, value) pair into a min-heap bt (Lines 21-24). If our temporary similarity score hashmap r already includes m sessions, we must choose whether to eliminate the oldest session. Consequently, we first extract the oldest session and its associated timestamp from the heap bt (Line 26). Should the current historical session j be more recent than the oldest session, it is necessary to exclude the oldest session from our temporary similarity score hashmap r and heap bt, subsequently updating both with the values from the current historical session j (Lines 27-31). Ultimately, we get the top-k scored sessions from the max-heap Ns inside the top-k similarity loop and return them (Line 33).

SERENADE:

We present the design and implementation of our scalable recom- mender system *Serenade*, which leverages VMIS-kNN (Section 3) and provides recommendations on the product detail pages of bol.com.

Design Considerations:

At the core of the design of our production system are two questions: (i) How to maintain the session similarity index over time; and (ii) How to efficiently serve next-item recommendations with low latency?

Index maintenance. We execute the index computation in an offiine manner once per day with a data-parallel implementation of the relational operations required for the index generation. This batch job is easy to schedule and scale; note that Serenade will thus only see sessions for new items on the platform with a delay of one day. This “cold-start” issue is no problem in practice however,because our e-commerce platform has a separate, specialised system for presenting new and trending items to users.

Low latency serving of next-item recommendations. The biggest challenge in our system is to serve session-based recom- mendations with a low latency for a catalog containing millions of items (our business constraint is to respond in 50 ms or less for at least 90% of all requests). As discussed in Section 1, we cannot precompute the recommendations due to the exponentially large input space of potential sessions, and we cannot apply approximate nearest neighor search techniques because our similarity function is not a metric. As a consequence, our recommendation servers have to be stateful, by maintaining copies of the evolving sessions, to be able to compute recommendations online on request. We decide to replicate our session index to all recommendation servers, and colocate the session storage with the update and recommendation requests, so that we only have to use machine-local reads and writes for maintaining sessions and computing recommendations. Note that similar techniques are often used to accelerate joins [16].

Execution

Figure 1 depicts the high-level architecture of

Serenade, based on the design considerations outlined in Section 4.1. Serenade has two elements: The offline component (depicted on the left side of the diagram) constructs the session index using click data and is implemented as an Apache Spark pipeline. The online component (depicted on the right side of the picture) calculates and provides session-based suggestions using VMIS-kNN, and is executed as a REST application. Serenade is built using the pre-existing Google Cloud infrastructure leased by bol.com.

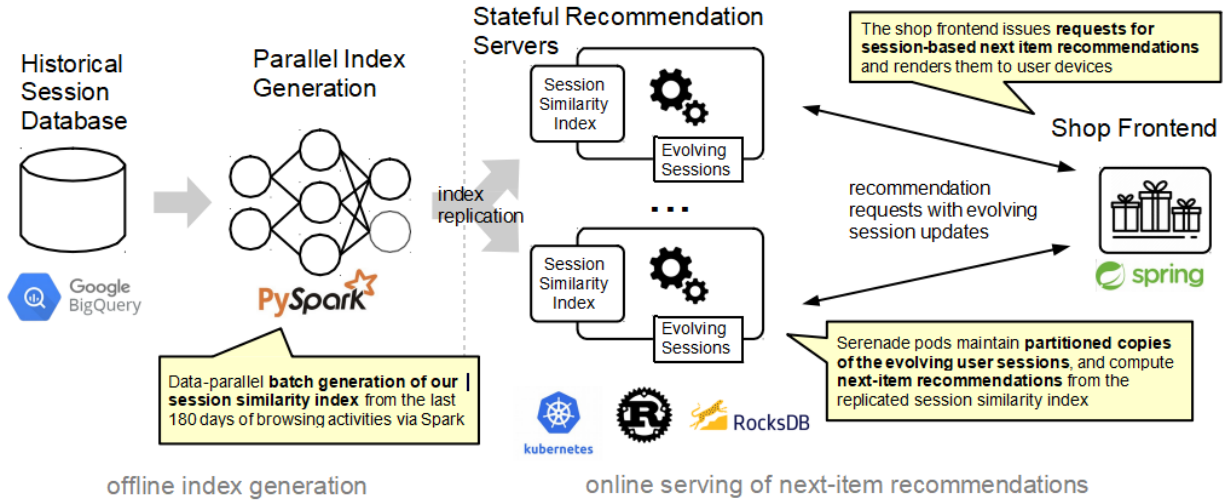

Figure 1: High level architecture of the Serenade recommendation system. The offline component (left) generates a session similarity index ❶ from several billion historical click events via a parallel Spark job in regular intervals. The online serving machines (right) maintain state about the evolving user sessions ❷ and leverage the session similarity index to compute next item recommendations with VMIS-kNN in response to recommendation requests from the shopping frontend ❸

Co-location of dynamic sessions and session modifications. As outlined in Section 4.1, it is necessary to colocate the developing sessions with the recommendation requests and session updates to compute current suggestions with minimal delay. We preserve the dynamic sessions in a local key-value store (RocksDB [6]) directly on the serving computers to eliminate supplementary network reads and writes. For colocation, it is necessary to split both the developing sessions and the recommendation requests, which include session updates, among the serving servers according to their session identifiers. To ensure that all update or recommendation requests for a certain session are consistently managed by the same machine, we implement request routing via "sticky sessions" using Kubernetes' session affinity feature [4]. Communication with RocksDB is very fast; in a microbenchmark including 10 million operations for our workload, the 99th percentile of read latency was measured at 5 microseconds, while the 99th percentile of write latency was recorded at 18 microseconds. This colocation strategy significantly improves latency compared to network reads and writes to a distributed key-value store such as BigTable, where the response latency for lookups is around 15ms at the 99.5 percentile based on our observations.

EXPERIMENTAL EVALUATION:

In the following, we first evaluate the prediction quality and index design of VMIS-kNN in Section 5.1, and subsequently evaluate the scalability and business performance of Serenade in offiine experiments and an online A/B test (Section 5.2). We provide the code for our experiments at https://github.com/bolcom/serenade-experiments-sigmod.Data sets. We use a mix of public and proprietary 20, 50, 500 1,000 10,000 20 50 500 1,000 10,000 20 50 500 1,000 10,000 20 50 500 1,000 to 10,000 Select datasets from e-commerce for our offline studies. We Conduct an experiment with the publicly accessible datasets RetailRocket, an e-commerce dataset from the business "Retail Rocket," and RSC15, a dataset used in the 2015 ACM RecSys Challenge, which are often utilized in comparative analyses of session-based recommendation systems. Furthermore, we generate the non-public datasets ecom-1m, ecom-60m, ecom-

90m, and ecom-180m by sampling data from our e-commerce platform, with progressively larger click counts. The statistics of these datasets are shown in Table 1. Each dataset has tuples representing the session_id, item_id, and timestamp of a click event on the platform.

Our unique dataset ecom-180m exceeds the size of the greatest publicly accessible dataset rsc15 by more than sixfold. We also include information on the distribution of clicks each session, including the 25th, 50th, 75th, and 99th percentiles. The majority of sessions on e-commerce platforms are notably brief, with the median number of clicks each session being less than five, and these numbers exhibit remarkable consistency across all six datasets. In the tail, the sessions from our platform are about double the duration of those in the public datasets (e.g., the 99th percentile is around 38 clicks in our data and 19 clicks in the public datasets).

	retailr	*rsc15*	*ecom-1m*	*ecom-60m*	*ecom-90m*	*ecom-180m*
clicks	86.655	31.708.401	1.152.458	67.017.567	89.885.761	189.517.566
sessions	23,318	7,981,581	214,490	10,679,757	13,799,762	28,824,487
items	21,276	37,483	110,988	1,760,602	2,263,670	3,305,412
days	10	181	30	29	91	91
public?	yes	yes	no	no	no	no
clicks per session						
p25	2	2	2	2	2	2
p50	2	3	4	4	4	4
p75	4	4	6	7	7	7
p99	19	19	28	36	38	39

Table 1: Public and proprietary datasets for evaluation.

VMIS-kNN

State-of-the-Art Prediction Quality. Before evaluating systems- related aspects, we run a sanity check experiment for the predictive performance of VMIS-kNN. We aim to confirm that VMIS-kNN also outperforms current neural-network based approaches in the task of session-based recommendation in e-commerce (as recently shown for VS-kNN [24, 30]).

Experimental setup. We replicate the setup from [24, 30], and compare the predictive performance of VMIS-kNN against three recent neural network-based approaches to session-based recom- mendation (GRU4Rec [20], NARM [27] and STAMP [29]) on various clickstream datasets sampled from our e-commerce platform. We create five versions of the *ecom-1m* dataset by sampling a million clicks from certain months in the past as historical sessions, and measure the prediction quality of the top 20 recommended items for each session of the subsequent day.

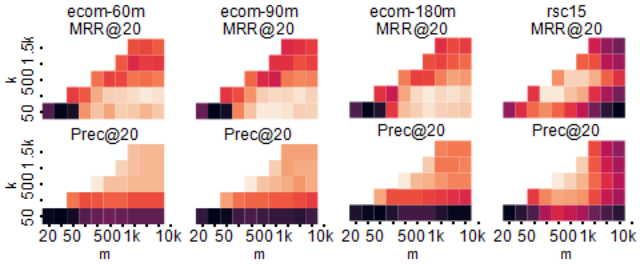

Figure 2: Sensitivity of MRR@20 and Prec@20 to the hyper-parameters □(the number of neighbors) and □(the number of most recent sessions per item) in our proprietary datasets.

We optimise the hyperparameters of each approach on samples of the training data, and report the average for each metric over all our evaluation datasets. We report the metric values averaged over all five versions of *ecom-1m*.

LEARNINGS & CONCLUSION:

We introduced our closest neighbor methodology VMIS-kNN, along with the design and execution of our scalable session-based recommendation system, Serenade. We performed a comprehensive offline assessment of VMIS-kNN and Serenade to substantiate our design choices, presenting detailed results on latency, throughput, and predictive efficacy of our recommender system derived from an online A/B test accommodating up to 600 requests per second for 6.5 million unique items across over 45 million user sessions on bol.com's e-commerce platform. Furthermore, we like to emphasize Serenade's minimal operating expenses, in addition to the donations outlined in Section 1: We operate two instances, each with three cores, in the Google Cloud (utilizing shared core n1-standard-16 instances) for the serving pods. Additionally, we necessitate 40 minutes on 75 n1-highmem-8 machines to generate the index with Spark daily, culminating in a total operational expenditure of under 30 euros per day for Serenade. As mentioned in Section 5.2.3, Serenade utilizes just one of the three cores on each instance, while the other cores are provisioned to accommodate high demands, such as during denial-of-service assaults. This cheap cost is particularly appealing when juxtaposed with the substantial expense of training deep learning models.

A neural learning-to-rank model on our platform incurs at least tenfold higher operational costs everyday and necessitates GPU processors for training, which are often a scarce resource in the cloud. In further research, we want to investigate the feasibility of executing our similarity calculations on a compressed index and the potential for progressively maintaining the index using a system like Differential Dataflow [32]. Acknowledgments. This research received funding from Ahold Delhaize. All material reflects the writers' opinions, which may not be shared or approved by their respective employers and/or sponsors.

REFERENCES:

[1] 2021. Actix Web. https://actix.rs.

[2] 2021. d-ary heap. https://docs.rs/dary_heap/0.3.0/dary_heap/.

[3] 2021. Istio sidecars. https://istio.io/latest/docs/reference/config/networking/ sidecar/.

[4] 2021. Kubernetes networking services. https://kubernetes.io/docs/concepts/ services-networking/service/.

[5] 2021. Performance Comparison of Neural and Non-Neural Approaches to Session- based Recommendation - Additional Information. https://rn5l.github.io/session- rec/.

[6] 2021. RocksDB. https://rocksdb.org.

[7] 2021. VS-kNN reference implementation. https://github.com/rn5l/session-rec/ blob/master/algorithms/knn/vsknn.py.

[8] Xavier Amatriain. 2012. Building Industrial-scale Real-world Recommender Systems. *RecSys* (2012), 7–8.

[9] Ioannis Arapakis, Xiao Bai, and B Barla Cambazoglu. 2014. Impact of response latency on user behavior in web search. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. 103–112.

[10] Badrish Chandramouli, Justin J Levandoski, Ahmed Eldawy, and Mohamed F Mokbel. 2011. StreamRec: a real-time recommender system. *SIGMOD* (2011), 1243–1246.

[11] Tong Chen, Hongzhi Yin, Hongxu Chen, Rui Yan, Quoc Viet Hung Nguyen, and Xue Li. 2019. Air: Attentional intention-aware recommender systems. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 304–315.

[12] Kyung-Jae Cho, Yeon-Chang Lee, Kyungsik Han, Jaeho Choi, and Sang-Wook Kim. 2019. No, that's not my feedback: TV show recommendation using watchable interval. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 316–327.

[13] Abhinandan S Das, Mayur Datar, Ashutosh Garg, and Shyam Rajaram. 2007. Google news personalization: scalable online collaborative filtering. *WWW* (2007), 271–280.

[14] James Davidson, Benjamin Liebald, Junning Liu, Palash Nandy, Taylor Van Vleet, Ullas Gargi, Sujoy Gupta, Yu He, Mike Lambert, Blake Livingston, and Dasarathi Sampath. 2010. The YouTube video recommendation system. *RecSys* (2010), 293–296.

[15] Ted Dunning and Ellen Friedman. 2014. *Practical Machine Learning: Innovations in Recommendation*. " O'Reilly Media, Inc.".

[16] Mohamed Y Eltabakh, Yuanyuan Tian, Fatma Özcan, Rainer Gemulla, Aljoscha Krettek, and John McPherson. 2011. CoHadoop: flexible data placement and its exploitation in Hadoop. *Proceedings of the VLDB Endowment* 4, 9 (2011), 575–585.

[17] Chen Gao, Xiangnan He, Dahua Gan, Xiangning Chen, Fuli Feng, Yong Li, Tat-Seng Chua, and Depeng Jin. 2019. Neural multi-task recommendation from multi-behavior data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 1554–1557.

[18] Lei Guo, Hongzhi Yin, Qinyong Wang, Bin Cui, Zi Huang, and Lizhen Cui. 2020. Group recommendation with latent voting mechanism. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 121–132.

[19] Jiayuan He, Jianzhong Qi, and Kotagiri Ramamohanarao. 2019. A joint context-aware embedding for trip recommendations. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 292–303.

[20] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2015. Session-based recommendations with recurrent neural networks. *arXiv:1511.06939* (2015).

[21] Haoji Hu, Xiangnan He, Jinyang Gao, and Zhi-Li Zhang. 2020. Modeling personal- ized item frequency information for next-basket recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1071–1080.

[22] Yanxiang Huang, Bin Cui, Wenyu Zhang, Jie Jiang, and Ying Xu. 2015. TencentRec: Real-time Stream Recommendation in Practice. *SIGMOD* (2015), 227–238.

[23] Dietmar Jannach and Malte Ludewig. 2017. When recurrent neural networks meet the neighborhood for session-based recommendation. *RecSys* (2017), 306–310.

[24] Barrie Kersbergen and Sebastian Schelter. 2021. Learnings from a Retail Recommendation System on Billions of Interactions at bol.com. *ICDE* (2021).

[25] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).