Expanding Boundaries in Scalable Session-Based Recommendations

Barrie Kersbergen

Expanding Boundaries in Scalable Session-Based Recommendations

Barrie Kersbergen

Expanding Boundaries in Scalable Session-Based Recommendations

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. dr. ir. P.P.C.C. Verbeek
ten overstaan van een door het College voor Promoties ingestelde
commissie, in het openbaar te verdedigen in
de Agnietenkapel
op maandag 7 juli 2025, te 17:00 uur

door

Barrie Kersbergen

geboren te Gorinchem

Promotiecommissie

Promotores:	prof. dr. M. de Rijke	Universiteit van Amsterdam
	prof. dr. S. Schelter	Technische Universität Berlin
Overige leden:	prof. dr. P.T. Groth	Universiteit van Amsterdam
	dr. H. Harmouch	Universiteit van Amsterdam
	dr. R.M. Jagerman	Google
	prof. dr. D. Jannach	University of Klagenfurt
	prof. dr. E. Kanoulas	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

This research was supported by and carried out at Bol and (partially) funded by Ahold Delhaize, through AIRLab.

Cover by Franka Kersbergen
Printed by Ridderprint, Alblasserdam

ISBN: 978-94-6522-114-4

Acknowledgements

You are holding my thesis, the results of a journey that has shaped me in many ways. This adventure has not only sparked innovation in my work at Bol but also enriched my academic skills – something I am incredibly thankful for. This research is the product of a collective effort, and I would like to take a moment to express my gratitude to everyone who played a role in making my PhD research a reality. None of this would have been possible without the support, insight and encouragement of many people and organizations.

First and foremost I want to express my heartfelt gratitude to my supervisors Maarten de Rijke and Sebastian Schelter, for their unwavering support throughout my PhD journey. Their expertise, exceptional guidance and contagious enthusiasm have been invaluable. Together, they created the perfect environment and opportunities for me to grow and thrive as a researcher, for which I am deeply grateful.

Maarten, your dedicated support made it possible for me to embark on this PhD journey as an external candidate. Your dedication, combined with a touch of humor, has been a true gift – one I will carry with me throughout my career. Sebastian, you were both a critical friend and a mentor, always challenging my ideas and providing thoughtful, multifaceted feedback. Your guidance was instrumental in raising the quality of my research to new heights. Thank you both so very much.

During my PhD research I had the unique opportunity to act as a *broker* between Ahold, specifically Bol, and the University of Amsterdam (UvA), represented by the Institute for Informatics. These organizations established a groundbreaking collaboration known as the A.I. in RetailLab (AIRLab), which is part of the Innovation Center for Artificial Intelligence (ICAI). I feel truly privileged to have been part of this venture. Bridging boundaries between different working environments unlocked new possibilities for collaboration, learning, and growth. In this way, the title of this thesis not only reflects the expansion of boundaries in recommendation systems, but also symbolizes the broader learning and development fostered within the AIRLab, which was central to my PhD research.

From Bol's perspective, I express my gratitude to Frans Muller from Ahold Delhaize as well as the directors and management team at Bol: Dagmara, Dennis, Erick, Ernst, Joris, Jurrie, Marcel, Mirko, Niels, Pim, Sven and Leon. Your dedication to stimulate innovation and bridging the gap between academia and industry has been both inspiring and key to the success of this research. Experimentation thrives on strong engineering and robust infrastructure, and I am deeply thankful to team RECO for their invaluable support. A special thanks goes out to An, Binyam, Bruno, Cuong, Danu, Gabor, Giles, Jelle, Joep, Karoliina, Marieke, Nian, Paola, Paulo, Sam, Sander and Sandra – your contributions have been essential and greatly appreciated.

From the perspective of the UvA, I would like to express my sincere gratitude to the directors and managers at the AIRLab: Bart, Evangelos, Maarten, and Sebastian. Your leadership, vision, and dedication to pushing the boundaries of cutting-edge research have been truly invaluable. Your commitment to fostering a vibrant and collaborative environment has greatly enriched my PhD journey and strengthened the impactful synergy between academia and industry. And of course, I would like to express my appreciation to all my colleagues at the AIRLab, the Information Retrieval Lab (IRLab)

and the DEEM lab from the Technische Universität Berlin. Thank you for making the labs such inspiring and enjoyable places to work. The thought-provoking discussions and collaborative energy have been a source of motivation and growth throughout this journey. I would also like to express my sincere gratitude to the co-authors of my work: Bojan, Frank, Olivier and Shubha. Your collaboration, valuable insights, and contributions have been instrumental in building our research. A shout-out thanks to Olivier, who co-authored three of my papers, thank you for your dedication and partnership. Lastly, thanks to the IRLab secretary for your support and for ensuring everything ran smoothly, allowing me to focus fully on my research. Working with all of you has been an enriching and rewarding experience, and I deeply appreciate the time and effort you invested in our work.

Paul Groth, Hazar Harmouch, Rolf Jagerman, Dietmar Jannach and Evangelos Kanoulas, it is a true honor to have you on my PhD committee. Thank you for your time to review my thesis. I deeply appreciate your involvement.

Lastly, I thank the people I hold closest to my heart – my family and friends – for their support and genuine interest in my work. Many thanks to Nicola and Frédéric, Machiel and Mariëlle, and especially to An, whose love, encouragement, and the countless joyful moments we continue to share have meant so much.

To Marjolijn, thank you for being my rock and always being there for me. Your love and support mean everything to me. You have an incredible ability to turn challenges into opportunities, making life brighter for everyone around you. I am endlessly grateful for your strength and vision – you are the steady force that keeps our family moving forward.

To my children, being part of your lives and watching you grow has been a constant source of inspiration, making this thesis even more meaningful. Franka, your sharp mind and effortless grace, combined with your good sense of humor and lighthearted touch, brighten the world. Your painted cover image adds a truly special element to this thesis. Walt, your boundless curiosity and excitement for learning and technology inspire me and everyone around you. Your warmth and sweetness light up every conversation.

This thesis is dedicated to my father-in-law whose appreciation of my achievements was something I deeply valued and will always remember. His kindness, generosity, and his remarkable ability for bringing people together has left a lasting impression on all of us. This work stands as a tribute to his memory and legacy. Thank you, dear Jos.

Barrie
Utrecht, December 2024

Contents

1 Introduction

Imagine that you own a massive online store with millions of products. A customer visits your website, and now you have to decide: What items are you going to show them? The goal is not only to guess what the customer might like, but also to show the right items at the right time, without overwhelming or boring your customer. For this challenge, recommendation systems have been developed [98] and play a vital role in helping users discover relevant items across various domains, including e-commerce [130], media [17], and social platforms [69]. Traditional recommendation approaches, such as trending products or category-based suggestions, often struggle to meet user needs in large catalogs. These approaches overwhelm users and miss individual preferences [74], highlighting the need for more personalized approaches using machine learning.

Although significant progress has been made in personalizing user experiences [42], challenges remain, particularly in scaling recommendations to massive product catalogs and user bases [30, 69, 130]. Many algorithm designers lack access to real-world systems and are limited to working with only small or moderately sized public datasets [25, 42]. Large industry deployments offer the unique opportunity to evaluate recommendation techniques on real users via A/B testing, a critical method for evaluating recommendation systems in real-world settings [20] to which most researchers do not have access. For these A/B tests to provide meaningful insights, it is essential that models can scale up to production setups like the large European e-commerce platform Bol, which requires handling billions of interactions. Achieving this level of scalability allows models to be tested under realistic conditions, where they must operate at reasonable training costs and within strict response times in the low millisecond range. Without this scalability, it remains unclear how effectively strong performance on small experimentation data will translate to performance in real-world production environments.

An important task within recommendation systems is Session-Based Recommendation (SBR), where these issues become especially critical. SBR systems are designed to predict the next item a user might interact with based solely on their actions during an individual session, a short, anonymous sequence of interactions on the site [25, 67, 93, 123, 125]. This type of recommendations is essential for settings where user preferences change quickly, and little historical data is available.

Modern e-commerce platforms, which feature millions of products and serve millions of users, rely heavily on recommendation systems to guide customers to relevant

items. However, the effectiveness of these systems is closely tied to the quality of the data on which they are trained. Data errors, such as the accidental recommendation of dangerous items [14] or issues arising from low-quality metadata [7], can severely degrade the recommendation performance. These errors are often unpredictable and typically only identified after they have negatively impacted the user experience. To mitigate these issues, data debugging techniques are essential. Recent advancements in data importance methods [32] aim to systematically highlight problematic data points, enabling more efficient identification and resolution of errors before they affect users. Unfortunately, given the massive scale of e-commerce datasets, the process of identifying the data errors that have the most significant negative impact, also called data debugging, can be prohibitively expensive.

Building scalable session-based recommendation systems poses unique challenges. First, precomputing recommendations for every possible combination of interactions is infeasible due to the sheer number of potential user sessions. Instead, predictions must be generated dynamically as users interact with the e-commerce platform. This requires high-performance systems that can efficiently process and respond to a large numbers of concurrent requests. Furthermore, these systems must adhere to strict service level agreements (SLAs), which define rigorous requirements for latency and specify the minimum throughput, measured in predictions per second, that the recommendation system must sustain. These SLAs are in place to prevent backpressure, resource bottlenecks, and delays that can propagate throughout the e-commerce platform, negatively affecting performance and user experience. Failing to meet the SLA will result in a failure to provide recommendations to users. Furthermore, such systems handle millions of sessions each day, yet each individual session involves only a small fraction of the product catalog, resulting in highly sparse data [76]. This presents a challenge in making accurate predictions with limited interaction data, yet it is essential for maintaining a high-quality user experience [67, 110].

In this thesis, we focus on expanding the boundaries of scalable session-based recommendation systems by addressing the challenges posed by large product catalogs, large- and sparse interaction data in e-commerce. Our work bridges the gap between academic research, which often lacks access to the scale required for real-world evaluation, and industry practices, where large-scale A/B testing is essential for refining and validating systems. We introduce novel methods that not only enhance the accuracy and the response latency of recommendations but also scale seamlessly to the demands of modern e-commerce platforms. We demonstrate that traditional nearest-neighbor approaches can match or even surpass the predictive performance of more complex neural network models on large-scale e-commerce datasets. Additionally, we explore the relationship between response latency and higher adoption rates of recommendations. These findings highlight the critical role of efficiency in recommendation systems and its strong correlation with user acceptance. To address these challenges, we introduce an indexing strategy that enables a state-of-the-art nearest neighbor method to efficiently manage billions of clicks. Our recommendation system requires low computational power, which is crucial for maintaining a low carbon footprint and promoting sustainability. A large-scale test confirms that its strong offline performance translates effectively to real-world environments, enhancing the acceptance of recommendations. Additionally, we develop a framework for the automatic evaluation of inference performance in neural

network-based session-based recommendation models, specifically tailored to declaratively specified deployment options. We also introduce a novel method designed for estimating Data Shapley Values for debugging click and purchase datasets containing millions of interactions, which are essential for recommendation systems.

By addressing these critical challenges, we expand the boundaries of session-based recommendation research, and provide solutions that are both theoretically sound and practically viable in large-scale contexts.

1.1 Research Outline and Questions

This thesis focuses on improving session-based recommendation systems in real-world e-commerce platforms by addressing challenges in predictive performance, response latency, resource efficiency, cost-efficient deployment and data quality issues.

Conducted at Bol, a large European e-commerce platform, this research addresses challenges posed by large product catalogs, billions of interactions and sparse data. We specifically investigate how to leverage and adapt recommendation algorithms within production constraints, aiming to balance computational demands with predictive accuracy and operational scalability.

Given the complexities of SBR systems in e-commerce, we have formulated the following research question:

RQ1 Which techniques can improve both the predictive performance and user acceptance of recommendations in large-scale SBR systems?

To address this question, we examine and compare neural and non-neural network approaches for SBR using proprietary data from Bol. We replicate a key study [70], where the algorithm called Vector-Session-kNN (VS-kNN) outperforms neural network-based methods in session-based recommendation on e-commerce data. Furthermore, we explore the impact of response latency on user acceptance through a large-scale A/B test conducted during a data center migration. This multi-faceted approach allows us to assess both predictive performance and user experience factors. To bridge the gap between identifying high-performing algorithms and deploying them effectively in production, we must address the challenges of scalability and responsiveness in a real-world setting. This leads us to our second research question:

RQ2 How can we scale VS-kNN to efficiently handle billions of interactions while maintaining low response times and adhering to production requirements in a real-world SBR system?

To address this question, we explore the development of a scalable variant of the VS-kNN algorithm that is designed to handle billions of interactions with millisecond response times and high throughput in a real-world session-based recommendation (SBR) system. This includes investigating the design and implementation of a production-ready, stateful recommendation system that meets production requirements, supports non-personalized recommendations for users without consent for personalization, and ensures resource-efficient deployment at scale. Additionally, we evaluate the system's

performance through both offline experiments and a large-scale A/B test conducted on Bol's product detail page to assess its effectiveness in a real-world setting.

Deploying session-based recommendation models in practice presents significant challenges for data scientists. While these models often achieve strong results in controlled research environments, translating them into production systems is far from straightforward. These challenges arise from the need to choose the most suitable model from a large number of models. Each model requires efficient online computation to generate recommendations on the fly. Many of these models are implemented in academic libraries, which often lack robust support for model deployment and inference optimizations. As a result, the inference performance and deployment costs of these models are often unclear. We are therefore interested in the following question:

RQ3 How can we automatically evaluate the inference performance of SBR models under different deployment options?

To address this question, we develop and utilize benchmarking frameworks capable of evaluating the inference performance of SBR models across various e-commerce deployment configurations. Additionally, we will explore how such frameworks can identify high-performing models and cost-efficient deployment setups tailored to diverse e-commerce use cases. This understanding is particularly crucial in large-scale e-commerce platforms like Bol, where millions of user interactions drive the generation of SBR, and the ability to fine-tune the deployment configuration is key to maintaining system performance and scalability.

However, these interactions often include problematic or noisy data, such as inconsistent clicks within a single user session. For instance, a session might involve a mix of clicks on wooden toys intended for vastly different age ranges, potentially degrading the quality of recommendations. Identifying and mitigating the impact of such harmful interactions is critical to maintaining the reliability and accuracy of production recommendation systems.

Quantifying the influence of individual data points is essential for improving recommendation quality in the presence of noisy or problematic interactions. Data Shapley values (DSV), grounded in cooperative game theory, offer a theoretically sound method for evaluating the contribution of each data point to the performance of a machine learning model. This approach offers valuable insights into which data points drive model predictions and which might be harmful. However, the practical adoption of Data Shapley values [32] in real-world systems is hindered by their computational complexity, which scales poorly with the size of modern datasets containing millions of interactions. Existing work has focused on small-scale experiments involving datasets with only hundreds of data points [32, 61], leaving a significant gap in scalability for real-world applications. This scalability challenge becomes even more critical in the context of large-scale production systems, such as the sequential kNN-based recommendation system deployed at Bol, which processes datasets containing millions of interactions daily. Addressing this challenge is both a theoretical and practical necessity, motivating our fourth research question:

RQ4 How can we efficiently compute Data Shapley values for sequential kNN-based recommendation systems on real-world datasets with millions of datapoints?

To address the computational challenges of estimating Data Shapley values for large-scale datasets, we investigate the design of a scalable approach tailored to sequential kNN-based models. This research focusses on leveraging key characteristics of interaction data, such as sparsity and locality, to optimize the estimation process. We explore methods for identifying and computing Shapley values for only the most relevant data points for each query while minimizing computations for less relevant interactions.

Through this investigation, we aim to develop a method that enables the practical application of Data Shapley values in real-world recommendation systems. This work will provide a foundation for understanding the relationship between data quality and system performance, contributing to the broader adoption of data importance metrics in large-scale recommendation systems.

Building on this, we explore whether Data Shapley values can be helpful for debugging real-world interaction data in sequential kNN-based recommendation systems. This motivates our final question:

RQ5 Are Data Shapley values helpful for debugging real-world interaction data in sequential kNN-based recommendation systems?

To assess the utility of Data Shapley values (DSVs), we investigate their potential as a tool for identifying and addressing inconsistencies or noise in real-world interaction data within sequential kNN-based recommendation systems. In large-scale e-commerce platforms like Bol, user interaction data often includes challenges such as irrelevant clicks, low-quality product metadata, or inconsistent user behavior, which can affect system performance.

We will explore the application of DSVs, which quantify the contribution of individual data points to the performance of machine learning models, to detect potentially problematic data. We will analyze DSVs across multiple datasets from Bol, including session-based browsing data and next-basket purchase histories, and examine their relationship to data issues that may not be evident through traditional data cleaning methods. This investigation will provide insights into the potential role of DSVs in debugging large-scale interaction datasets.

This concludes the overview of the research questions that are answered in this thesis. Next, we turn to an overview of the main contributions of this thesis.

1.2 Main Contributions

We divide our contributions into theoretical, empirical, and software contributions.

Theoretical contributions

- We introduce an algorithm for optimizing serving latency during bulk updates, which dynamically adjusts the insertion rate based on real-time CPU load monitoring. This algorithm ensures that the system adheres to latency SLAs while maintaining optimal CPU performance, preventing overload during large-scale data updates (Chapter 2).

- We introduce the Vector-Multiplication-Indexed Session kNN algorithm, which we abbreviate to VMIS-kNN, an index-based variant of a state-of-the-art nearest neighbor algorithm for session-based recommendation, which scales to use cases with hundreds of millions of clicks to search through (Chapter 3).

- We introduce an algorithm for backpressure-aware load generation that dynamically ramps up throughput while monitoring the system's ability to handle requests in real-time. This algorithm ensures graceful degradation under high load conditions, identifying throughput thresholds where models fail, and enabling more accurate latency measurement during stress testing of deployed models (Chapter 4).

- We introduce KMC-Shapley, a scalable variant of a recent Monte Carlo-based algorithm to estimate the Data Shapley Value [32] that exploits the fact that our recommendation systems are built on a special class of models, namely nearest-neighbor models for sequential recommendation, and that the interaction data in real-world recommendation systems is extremely sparse (Chapter 5).

Empirical contributions

- We conduct a study on enhancing the predictive performance of a real-world recommendation system in production at Bol. We confirm the finding from [70] that simple VS-KNN approach outperforms neural network based approaches in session-based recommendation on e-commerce data (Chapter 3).

- We discuss design decisions and implementation details of our production recommendation system Serenade, which applies stateful session-based recommendation with VMIS-kNN, and can handle more than 1,000 requests per second with a response latency of less than seven milliseconds in the 90th percentile using only 2 vCPU's (Chapter 3).

- To the best of our knowledge, we provide the first empirical evidence that the superior predictive performance of VMIS-kNN/VS-kNN from offline evaluations translates to superior performance in a real-world e-commerce setting; we find Serenade to drastically increase a business-specific engagement metric by several percent, compared to the legacy system at Bol (Chapter 3).

- We present an experimental study for ten different SBR models in challenging settings resembling real-world workloads encountered at Bol. We determine performant and cost-efficient deployment options in terms of model and cloud instance types for a variety of online shopping use-cases (Chapter 4).

- We identify severe performance bottlenecks in the open-source TorchServe inference server from PyTorch ecosystem and in the implementations of four models from the open-source RecBole library and submit bug reports [97]. These include the multiplication of very sparse matrices using dense operations in RepeatNET and repeated CPU-GPU data transfers in SR-GNN and GC-SAN (Chapter 4).

- We discuss the design of our library Illoominate for debugging large scale recommendation data via data valuation (Chapter 5).
- We demonstrate that our KMC-Shapley algorithm is orders of magnitude faster in estimating Shapley values compared to TMC-Shapley, specifically for sequential k-nearest neighbors (kNN) recommendation models on very sparse e-commerce datasets. The results highlight the efficiency and scalability of our approach, tested on both public and private datasets containing millions of interactions (Chapter 5).
- We discuss various applications of Illoominate on click and purchase data from Bol. These applications include the identification and removal of dangerous products and products with low quality metadata, the detection of account sharing behaviour and the improvement of ecological sustainability of recommendations via data pruning. Furthermore, we provide initial evidence that our computed DSV's are also meaningful for neural recommendation models (Chapter 5).

Software contributions

- We make our implementation of Serenade, the session-based recommendation system currently in production at Bol, available under an open license, including downloadable binaries for Linux, Mac and Windows, at `https://github.com/bolcom/serenade` (Chapter 3).
- We make the source code of Etude and our experimental results available under an open license at `https://github.com/bkersbergen/etudelib` (Chapter 4).
- We make our data importance framework Illoominate available under an open-source license, including a python binding `https://github.com/bkersbergen/illoominate` (Chapter 5).

1.3 Thesis Overview

This section briefly outlines the structure of the thesis and offers guidance on how to approach it. Chapters 2–5 address the research questions outlined earlier. Each chapter is self-contained, offering the necessary background and relevant related work. In Chapter 6, we conclude the thesis and suggest directions for future research.

1.4 Origins

The chapters in this thesis are all based on published papers.

Chapter 2 is based on the following publication:

> B. Kersbergen and S. Schelter. Learnings from a retail recommendation system on billions of interactions at bol.com. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2447–2452, 2021. doi: 10.1109/ICDE51399.2021.00277 [51].

BK: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Resources, Software, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing. SSC: Conceptualization, Methodology, Supervision, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing.

Chapter 3 is based on the following publication:

B. Kersbergen, O. Sprangers, and S. Schelter. Serenade – Low-latency session-based recommendation in e-commerce at scale. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 150–159, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392495. doi: 10.1145/3514221.3517901 [52].

BK: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing. OS: Formal Analysis, Investigation, Methodology, Software, Writing – Original Draft Preparation, Writing – Review & Editing. SSC: Conceptualization, Methodology, Software, Supervision, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing.

Chapter 4 is based on the following publication:

B. Kersbergen, O. Sprangers, F. Kootte, S. Guha, M. de Rijke, and S. Schelter. Etude – Evaluating the inference latency of session-based recommendation models at scale. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 5177–5183, Los Alamitos, CA, USA, May 2024. IEEE Computer Society. doi: 10.1109/ICDE60146.2024.00389 [53].

BK: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing. OS: Validation, Formal analysis, Writing – Original Draft Preparation, Writing – Review & Editing. FK: Data Curation, Investigation, Software, Validation. SG: Software. MdR: Conceptualization, Funding Acquisition, Methodology, Supervision, Writing – Review & Editing. SSC: Conceptualization, Methodology, Software, Supervision, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing.

Chapter 5 is based on the following publication:

B. Kersbergen, O. Sprangers, B. Karlaš, M. de Rijke, and S. Schelter. Illoominate – Scalable debugging of recommendation data in e-commerce. Under submission, 2025 [54].

BK: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing. OS: Validation. BKš: Validation, Formal analysis, Writing – Review & Editing. MdR:

Conceptualization, Methodology, Supervision, Writing – Review & Editing. SSC: Conceptualization, Investigation, Methodology, Software, Supervision, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing.

2

Learnings from a Retail Recommendation System on Billions of Interactions at bol.com

2.1 Introduction

Today's internet users face an ever increasing amount of information. This situation has triggered the development of recommendation systems: intelligent filters that learn about the users' preferences and suggest relevant information for them. With rapidly growing data sizes, the predictive performance, processing efficiency, and scalability of machine learning-based recommendations systems and their underlying computations becomes a major concern.

In this chapter, we describe the architecture of a real-world recommendation system ABO for bol.com, a large European e-commerce platform which handles billions of interactions on several dozen million items every day in Section 2.3. The ABO ("Anderen bekeken ook," Dutch for "others also viewed") recommendations are shown on the product detail page[1] to enable customers to discover other products that are relevant to them, such as different versions of the same product, similar products, or products that are complementary to the displayed item. We describe the individual components of our system, which are backed by cloud infrastructure from the Google Cloud Platform such as BigTable and BigQuery. In addition, we detail our nearest-neighbor-based recommendation approach, we discuss how we conduct distributed offline model training, and how we efficiently serve the recommendations online with low latency.

A natural question when operating a real-world recommendation system is how to improve its predictive performance. In this work, we explore two directions for improvement and present the results of two corresponding studies. This leads us to our first research question.

This chapter was published as B. Kersbergen and S. Schelter. Learnings from a retail recommendation system on billions of interactions at bol.com. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2447–2452, 2021. doi: 10.1109/ICDE51399.2021.00277.

[1] https://www.bol.com/nl/p/-/9300000032324896

RQ1 Which techniques can improve both the predictive performance and user acceptance of recommendations in large-scale SBR systems?

To address this question, we investigate the potential of *algorithmic improvements* in Section 2.4. Neural networks have shown outstanding performance in computer vision [58] and natural language processing tasks [9], and therefore we evaluate recently proposed neural network-based approaches [41, 63, 67, 125] for session-based recommendation on real data from our platform, based on an existing academic study [70]. We evaluate the predictive performance of these neural networks, as well as their deployability for production settings, in terms of training time, cost of hyperparameter search, prediction latency, and scalability.

We also focus on *system-specific improvement* by optimizing our serving infrastructure to drastically reduce its response latency (Section 2.5). We control the insertion rate of bulk updates into the production database of our recommendation system, in order to adhere to a latency SLA (service-level agreement) of 50ms for recommendation responses. A large-scale online A/B test on 19 million user sessions was conducted to investigate the impact of this response latency reduction on the predictive performance of our recommendation system.

In summary, we provide the following contributions in this chapter:

- We discuss the design of a large-scale recommendation system handling billions of interactions on a European e-commerce platform (Section 2.3).
- We present two studies on enhancing the predictive performance of this system: (i) We evaluate recent neural network-based approaches on proprietary data from our e-commerce platform, and confirm recent results outlining that the benefits of these methods with respect to predictive performance are limited, while they exhibit severe scalability bottlenecks (Section 2.4); (ii) We optimise the response latency of our serving system, and conduct an A/B test on the live platform with more than 19 million user sessions, which confirms that the latency reduction correlates with a significant increase in metrics based on purchases and revenue (Section 2.5).
- We discuss the implications of our findings with respect to real-world recommendation systems, as well as future research on session-based recommendation (Section 2.6)

2.2 Related Work

Recommendation systems are an active research area [94], which is regularly fueled by industry challenges such as the "Netflix Prize" competition [57]. Unfortunately, it is often difficult to translate academic progress into deployable solutions which need to be able to handle billions of interactions. The winning solution of the Netflix challenge for example never went into production [4]. A classical approach to recommendation mining are nearest-neighbor methods [13, 62, 98, 99], which are widely deployed in industry [19, 20, 44]. In the area of session-based recommendation, which is especially important for e-commerce scenarios, many neural network-based approaches have

recently been proposed [41, 63, 67, 125], and there is an ongoing discussion of their relation to conceptually simpler nearest-neighbor methods, which outperform them on many datasets [45, 70].

2.3 System Architecture

In this section we provide a high-level overview of the architecture of our recommendation system, as illustrated in Figure 2.1. We describe our recommendation approach (Section 2.3.2), distributed model training (Section 2.3.3), and how to serve low-latency recommendations in response to user requests (Section 2.3.4).

2.3.1 Overview

ABO is designed for distributed computation and implemented on top of Apache Spark [126]. All of our components are loosely coupled, so they can run independently from each other, while exchanging data via a distributed filesystem. Our architecture consists of a batch system that is responsible for executing data-intensive model training workloads offline on 18 billion user-item interactions which is 3.5TB in size. We periodically precompute the recommendations for every item in our catalog, typically once per day for all 70M products. The online serving system is responsible to serve recommendations online with low latency. The most important Big Query data source for the candidate algorithms is the click data containing the historical production interactions from customers, including clicks, purchases and shopping cart additions. This data source grows by about 30M records per day. We clean and filter this data, and join it with the catalog and offer data sources to determine active recommendable products. We describe the components of our system from Figure 2.1 in detail:

Candidate generation. We compute nine recommendation models that each generate recommendations for products in the catalog: (i) *Related-Products* recommends products that where clicked after clicking items, (ii) *Order-After-Click* recommends the products that where purchased after clicking an item, (iii) *Purchased-Together* recommends products that are purchased together, (iv) *Family* recommends products that only differ in one product attribute such as size or color from a recent item. (v) *Similar* recommends products that are similar in title and description using cosine similarity with tf-idf weighting. (vi) *Trending* recommends products that are popular in the same category (vii, viii, ix) *Creator*, *Brand* & *Publisher* recommend products from the same content creator, brand or publisher. Note that each model has it own MLlib pipeline and the parallel execution and machine resource allocation is automatically handled by Spark.

Ensembling & business rules. The ensembling component aggregates the outputs of all nine recommendation models to generate a final set of recommendations per item, and applies a set of manually defined business-specific filters to the recommendations. These filters remove potentially unwanted recommendations, such as combinations of adult and non-adult products.

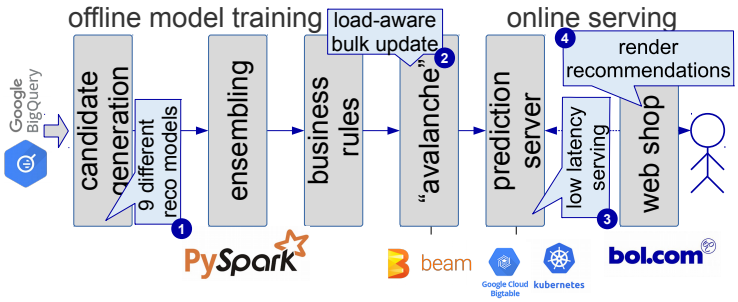


Figure 2.1: Architecture overview of the offline training and online serving components of our retail recommendation system, based on Apache Spark and Apache Beam, and backed by infrastructure from the Google Cloud Platform.

"Avalanche". This component handles the bulk update of the pre-computed recommendations into the Bigtable database of the prediction server, our online serving component. It is implemented on top of Apache Beam, and converts the recommendations to a Protobuf format with a fixed maximum amount of 21 recommended products to guarantee fast deserialization at serving time. The avalanche component will automatically adjust its insertion rate into BigTable to guarantee a low serving latency during the ingestion (see Section 2.5 for further details).

Prediction server. The task of this component is to serve the recommendations to end users with low latency. We implement it in Java on top of the Spring framework running in a Kubernetes cluster. This component elastically scales its underlying cluster of machines using Kubernetes's Horizontal Pod Autoscaler, by automatically spawning or removing extra serving nodes based on the overall CPU load.

Webshop. This component communicates with the prediction server and renders the final web page served to customers, which also includes the product recommendations.

2.3.2 Recommendation Approach

Next we describe the nearest neighbor-based algorithms underlying the first three approaches of our recommendation models, and subsequently discuss how to implement the model training in a scalable dataflow system.

Item-based recommendation. We leverage item-based collaborative filtering for recommendations [98], a simple but highly popular approach, deployed in many production settings [19, 20, 44]. This approach compares user interactions to find related items in the sense of "people who like this item also like these other items." The resulting pairs of cooccuring products are later on combined with the output from the other models and re-ranked. Our system bases its recommendations on so-called "implicit feedback data" e.g., count data that can easily be gathered by recording user actions such as clicks, shopping cart items or purchases.

Collection and scoring of item cooccurrences. We first clean the recorded interaction data in windows of 24 hours: we filter out data from users who visited an unusual

amount of items during that time window. Next, we set the item cooccurrence count $c^{(t)}_{uijg}$ between item i and item j to one if we find a cooccurrence in a sliding window of 42 days for a user u who interacted with these items in the given window t. Note that the count is 0 otherwise, and that we additionally record the type of interaction g (e.g., click or purchase). Note that we do not use personal identifiable information to determine the recommendations in order to respect the privacy of our users. Based on the collected cooccurrences, we compute the score s_{ij} for all observed cooccurrences $c^{(t)}_{uijg}$ of an item pair ij as follows. We sum up the observed cooccurrence counts across all users u, windows t and interaction types g. We apply an interaction specific weight w_g to each cooccurrence and decay the score with a function $\gamma(t)$ based on the amount of days passed since that interaction happened, to compute the final similarity score $s_{ij} = \sum_u \sum_t \sum_g \gamma(t)\ w_g\ c^{(t)}_{uijg}$. We filter out item pairs below a certain threshold to prevent recommending pairs with low user support.

Ensembling. We finally compute an ensemble to aggregate the item-to-item recommendations from our nine different models. Our models include collaborative filtering based approaches like the one described previously, but also content-based approaches that compute item similarity based on item metadata. The latter approach has the advantage to also provide recommendations for 'cold-start' items for which we have not seen interactions yet and which cannot be handled by out-of-the-box collaborative filtering algorithms therefore. The ensemble combines the algorithm-specific scores from all models with a weighted sum with manually tuned chosen weights. The final weights are based on (i) the results of offline evaluation experiments, measuring the normalized discounted cumulative gain (NDCG) on held-out conversion, revenue and click data; (ii) online A/B tests in the live system measuring several business metrics; and (iii) qualitative offline evaluation by business experts.

2.3.3 Distributed Model Training

Next, we describe how to compute our cooccurrence-based recommendations with Apache Spark [126]. The input for our model is clickstream data that spans several years of time, containing more than 18 billion user-item interactions at the time of writing. The dataset comprises 3.5 terabytes of data, and is stored in Google BigQuery (BQ). This data contains historical user actions from our webshop such as product views, additions to the shopping cart, and purchase events.

We process this data in parallel with Apache Spark. We model our computations based on the abstractions for feature transformations, models and evaluators of Spark MLLib [77], which improve the reusability and composibility of the computations. In accordance with our previously described recommendation approach, we partition the user-item interaction data by day (corresponding to the window t from the previous section), and collect the item cooccurrences via a distributed self-join on the user id. The collection of the item cooccurrences for one day is independent of the collection of the item cooccurrences for other days which allows for massive parallelism in the computation. We store the respective cooccurrence counts per day in the distributed file system. The scoring of the cooccurrences is conducted by a second Spark job, which reads these cooccurrence counts, sums them up according to the scoring function and

retains the top scored item pairs per item.

We execute the resulting computation in the Google cloud leveraging the Dataproc service.[2] The corresponding cluster is configured to autoscale up to 75 worker nodes of type `n1-highmem-8` each with a 500GB disk. We use dataproc image_version '1.4' which provides Apache Spark '2.4'. Executing the Spark jobs for the offline model training (all nine recommendations models, ensembling, and business rule filtering) takes approximately two hours.

2.3.4 Online Serving

The concerns of generating and serving the recommendations are separated in our architecture. The prediction server component is responsible for serving the recommendations with low latency. We implement it as a Java Spring service with BigTable as database backend. Our service is designed to perform auto-scaling of its computational resources: it spawns or revokes additional machines based on the current CPU load, and is also responsible for managing the amount of BigTable machines. The service achieves this by monitoring the CPU load of the BigTable machines every minute, and adjusting the amount of machines correspondingly. In order to prevent stale data being served from BigTable, we apply a simple optimistic locking scheme: After data is written to BigTable, we update a counter in a BigTable table and set the timestamp to the start of the insertion time. Our service compares the timestamp of that counter value with the column value timestamp before serving the value for that key. If the counter timestamp value is greater than the timestamp of the column value the row is not returned. Our webshop performs approximately 1,500 requests per second to the serving component of which approximately 400 requests per second are going to the recommendation system.

A natural question when operating such a real world recommendation system is how to improve its predictive performance. In the following sections, we explore two directions for improvement.

2.4 Neural Networks versus Nearest Neighbor Methods for Session-Based Recommendation

In recent years, neural networks have drastically outperformed traditional ML models in various domains such as computer vision [58] and natural language processing [9]. It is therefore a natural question to explore the benefits of neural-based approaches in comparison to nearest neighbor techniques (such as our system) for our e-commerce scenario as well. In contrast to academic studies, we have to look at additional dimensions such as scalability and training cost as well, in order to judge whether it would make sense to deploy a neural-based approach.

The academic setup that is closest to our production use case is *session-based recommendation*, where the goal is to predict the next item (or the set of next items) that a user will interact with, given the current items of her session on e-commerce

[2] https://cloud.google.com/dataproc

datasets. Interestingly, recent academic research [45, 70] indicates that neural-based approaches do not outperform classical nearest neighbor approaches in this scenario. We replicate a prominent study [70] on a sample of our production data to evaluate whether it may be beneficial to invest in neural approaches for our use case, and to investigate the scalability and training performance of current neural-based approaches for session-based recommendation on a large-scale e-commerce data. We include our approach as well, even though it has not been specifically designed and optimised for this specific type of evaluation, but can be considered a special case of session-based recommendation as it only considers the most recent item in the session. Our main goal of this study is to confirm that the family of nearest neighbor-based algorithms provides state-of-the-art performance on e-commerce recommendation tasks.

2.4.1 Data & Algorithms

Dataset. We use real-world clickstream data from our platform for our study. We create five samples, each spanning 31 days from different times of the year, in order to minimise the impact of seasonal effects. We bin users by the amount of purchases that they made, and apply stratified sampling based on these bins to select a set of sessions that represents the activities of a wide range of our customers. We include around 1.2 million actions on 120 thousand items in each sample.

Algorithms. We include eight recommendation algorithms in our evaluation, in accordance with [70]. Four of these employ neural network-based learning approaches for session-based recommendation: GRU4Rec [41], an RNN-based approach in combination with ranking loss functions [40] tailored to the session-based recommendation setting; NARM [63], an attention mechanism-based approach that aims to learn a user's sequential behavior in the current session; STAMP [67], an attention mechanism-based approach that aims to learn a user's sequential behavior in the current session by also learning the priority of the last item; and NEXTITNET [125], an approach employing convolutions to learn high-level representations of both short-and long-term item dependencies.

We additionally include the following four nearest-neighbor methods from [70] in our study: AR *(Association rules)*, an approach based on counting pairwise item cooccurrences in the observed sessions; SR *(Sequential rules)*, a frequent pattern mining approach based on sequential cooccurrences in the observed sessions, as well as S-KNN & VS-KNN *(Session k-Nearest Neighbor)*, two nearest-neighbor approaches based on session similarity, where the latter puts more emphasis on recent events in sessions.

Experimental setup. We evaluate the predictive performance of all methods on our five data samples, based on the experimentation framework provided by Ludewig et al. [70]. We use the first thirty days of each dataset for training and evaluate the predictions for the subsequent 31st day. Testing on a consecutive day also resembles our production setting, where we re-train our model every day. For each session in the test data we replay one interaction after another. After each revealed interaction, we compute recommended items and compare them to the remaining interactions. We execute the training for each model in the Google cloud on a `n1-highmem-8` instance

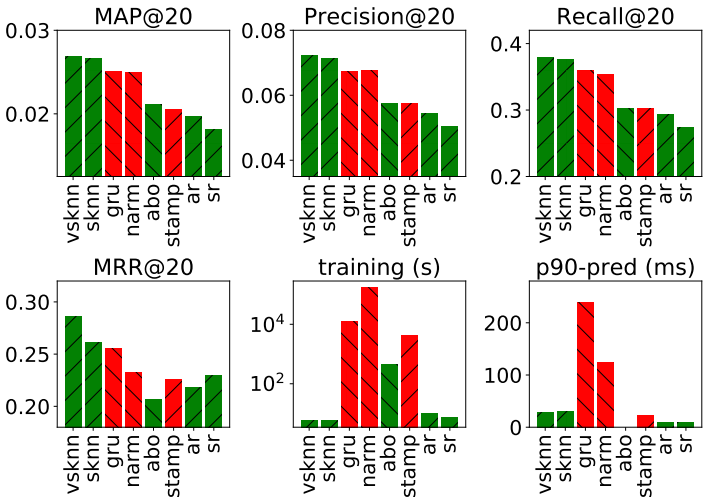


Figure 2.2: Prediction quality, training time and the 90th percentile of the prediction time for session-based recommendation with the neural-based (red) and nearest-neighbor-based (green) recommendation approaches (including our approach ABO which was not designed for this task), averaged over five different data samples from our e-commerce platform.

with a `nvidia-tesla-t4` GPU.

Metrics. We report four metrics computed from the top 20 recommended items: Mean Average Precision (MAP@20), Precision (P@20) and Recall (R@20) evaluate to what extent an algorithm is able to predict the next items in a session, while Mean Reciprocal Rank (MRR@20) evaluates to what extent an algorithm is able to predict the immediate next item in a session. We also do not report prediction times for ABO, which are hard to compare to the times of the other approaches, as ABO executes key-value lookups in BigTable in GCP, which includes network communication.

Hyperparameter optimization. Hyperparameter search for neural-based recommendation methods can be very time consuming, even on small data [70]. We therefore apply the following approach to tune the hyperparameters of all methods. We explore 100 combinations of hyperparameters with a random search on a data subset comprised of 100,000 interactions, and select the hyperparameters that result in the best MRR@20. Note that we had to schedule the hyperparameter search in parallel on different machines in the cloud to retrieve results in a reasonable time, even when allowing the methods to use GPUs. The hyperparameter search for NARM for example took more than six days of compute time, (in contrast to less than two hours for the VS-KNN approach)!

Results. Figure 2.2 shows the results of the predictive performance measured by our metrics, and the train/test time for neural-based and nearest-neighbor based approaches averaged over our five different data samples. We do not provide results for the NEXTITNET approach because it repeatedly crashed during training, which we attribute

to a dependency issue with the provided implementation.

We find that the nearest neighbor approach VS-KNN consistently outperforms all neural-based approaches, and even provides a higher evaluation score for MRR@20, the metric for which the neural-based approaches have been designed. S-KNN outperforms all three neural networks in four out of five metrics as well. ABO outperforms the neural-based method STAMP in MAP and Precision, as well as the baseline methods AR and SR (even though it has not been designed and optimised for the task of session-based recommendation). The time required to train the neural approaches is at least an order of magnitude larger than the training time of the neighbor based approaches. Additionally, two of the three neural based methods require much more time for inference than the nearest neighbor methods. The 90th percentile of the time needed for a prediction with GRU4REC and NARM on a small dataset is already higher than 100 milliseconds.

Discussion. The experimental results on our proprietary data confirm the findings from the original study [70]: simple nearest neighbor methods outperform recent neural network based approaches for session-based recommendation on e-commerce data. In addition, we make several observations with respect to the deployability of the neural methods in a real-world production scenario: (i) These methods exhibit extremely long training times for hyperparameter search and model training even on very small datasets (100k observations for hyperparameter search and 1M observations for training). It is unclear whether they would even scale to our current production workload of several billion interactions at a reasonable training cost and time; (ii) In addition, we saw the time to produce a recommendation with GRU4REC and NARM on our small evaluation dataset is already in a range that is far from usable in a real-world serving system, which has to guarantee response times in the low millisecond range.

2.5 The Impact of Serving Latency on Recommendation Performance

Most research focuses on improving the predictive performance of recommendation systems via algorithmic changes. Motivated by our production setup, we are interested in the impact of orthogonal, systems-related improvements. These are in general hard to study for academic researchers without access to real world systems. Work from [5, 56] indicates that a reduction of the response latency has a positive impact on the acceptance rate of a search engine. Therefore, we decide to investigate the impact of response latency on our recommendation system as well.

Data center migration. As part of a bigger reorganization of infrastructure we migrated our serving component and its database from our proprietary data center (DC) to the Google Cloud Platform (GCP). This migration included several changes such as upgrading the version of Java and the libraries we use in our serving component, as well as rewriting code for retrieving records from the database and serving them. We use BigTable in GCP as alternative to Apache Cassandra from the DC setup. In the DC, we ran the serving component in VMWare, while in GCP we run it via Docker images. Note that the serving component has to adhere to a strict service level agreement in each setup: If the webshop does not receive a response within 150ms it will discard the

Algorithm 1 Control of the insertion rate of Avalanche.

function CONTROL_INSERTION_RATE(r, c, b, d)
  Input: Insertion rate r (insertions per sec) conducted by Avalanche,
    average CPU load in BigTable c (percent), number of BigTable nodes b,
    number of Dataflow nodes d in Avalanche.
  Output: Updated insertion rate limit for Avalanche.
  $r_{\text{max}} \leftarrow (b \cdot 10000) \,/\, d$
  $\delta_{\text{cpu}} \leftarrow 40 - c$
  if $c \geq 40$ **return** $\max(1, r + 20 \cdot \delta_{\text{cpu}})$
  elseif $c \geq 35$ **return** r
  elseif $c \geq 30$ **return** $\min(r_{\text{max}}, r + \delta_{\text{cpu}})$
  else return $\min(r_{\text{max}}, r + 3 \cdot \delta_{\text{cpu}})$

request and render the web page without the recommendation.

Optimization for serving latency during bulk updates. We notice that this SLA is likely to be violated during the daily bulk update of our pre-computed recommendations in the prediction server (Section 2.3). In order to adhere to the latency SLA, the recommendations must be inserted without raising the CPU load on BigTable too much. This is challenging because the CPU load varies during the day and even while writing the data. We address this issue by sharing the responsibility of maintaining a low BigTable CPU load with the Avalanche job, which inserts updated recommendations into BigTable. We add a rate limiter mechanism to control the insert rate of Avalanche as shown in Algorithm 1, aiming for an average CPU load of 35% in the BigTable nodes. We obtain the CPU load of the BigTable nodes every minute, and update the insertion rate of the Apache Beam job in Avalanche which conducts the bulk update. Note that the number 10,000 refers to the minimum amount of requests per second that a BigTable machine is guaranteed to answer.

2.5.1 Experimental Evaluation

As part of our infrastructure migration, we conduct a large-scale online A/B test on our e-commerce platform to measure the impact of the reduction in serving latency on the acceptance of our recommendations.

Experimental Setup. Our website shows recommendations on every product page and makes them available as soon as the page loads. We run a large-scale online A/B test for a period of two weeks, where we divide visitors into two groups: The first group is served from the DC architecture, while the second group is served from the GCP architecture. We ensure that each visitor remains in the same A/B test group for the duration of the experiment via a persistent cookie that stores the group assignment. In total, we include more than 19 million user sessions in this experiment. The recommendations served from the DC and GCP architectures originate from the same algorithm and data, and are therefore identical. We measure the distribution of the latency between our prediction servers and webshops (deployed in DC and GCP), as well as two important business metrics based on orders and revenues. The circuit-breaker timeout in webshop for the recommendation service had been set to 150ms for this experiment, and our infrastructure is constantly monitored for request discards by site reliability teams.

Results & Discussion. During the period of the experiment, we observe that the 90th percentile of the response latency distribution between the prediction service and our webshop[3] running in the DC is 32ms, while the GCP setup only requires 15ms. Furthermore, we observe a 2.19% increase in an important order-based metric, as well as a 2.31% increase in a corresponding revenue-based metric. We confirm that the resulting positive impact in metrics is statistically significant with a chi-squared test of independence. In summary, we find that important order and revenue based metrics correlate positively with a reduction in response latency by 17ms. Our results indicate that users are more likely to perform a purchase if recommendations are served with lower latency, given two content-wise identical recommendation systems.

2.6 Learnings & Future Work

In this chapter, we focus on how to improve the predictive performance of a real-world recommendation system with both algorithmic and systems-related approaches. We confirm the finding from [70] that the simple nearest-neighbor-based VS-KNN approach outperforms modern neural network-based methods in session-based recommendation on e-commerce data. We additionally found that VS-KNN is orders of magnitude faster to train than the neural-based methods. This is an important property for a production systems that have to conduct regular retraining of their models, and adhere to strict time constraints for that.

Our second study indicates that reducing the response latency for serving has a significant impact on the acceptance of recommendations in e-commerce. We A/B tested this impact on over 19 million sessions where we were able to provide visitors a better experience by improving the serving latency, which resulted in a significant increase in business-relevant metrics. We ran this study during an ongoing data center migration, which gave us the unique opportunity to investigate the effect of improving the latency, instead of making it artificially worse, as done in previous studies for search engines [5, 56].

In the future, we will explore how to scale up well-scoring algorithms for session-based recommendation (in particular VS-KNN) to a full production workload with several billion interactions. We think that our studies also outlined interesting research directions for improving the suitability of the neural-based recommendation algorithms for production settings.

In the next chapter, we will explore adaptations to VS-kNN for efficient recommendations computation at scale and present the design and implementation of our scalable session-based recommendation system.

[3] Note that we measure the response latency at the webshop, our frontend server; the total latency for the customers includes additional network communication and the rendering of the web page on their devices.

3

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

In this chapter, we explore how to adapt the VS-kNN algorithm to enable efficient recommendation computation at scale, addressing the challenges posed by large datasets and the need for low-latency responses in real-world applications, introducing our adaptation of VS-kNN: Vector-Multiplication-Indexed-Session kNN (VMIS-kNN). This leads to research question:

RQ2 How can we scale VS-kNN to efficiently handle billions of interactions while maintaining low response times and adhering to production requirements in a real-world SBR system?

We further detail the design of our production system, Serenade, built on VMIS-kNN to meet the demands of large-scale recommendations. Serenade achieves high efficiency by collocating session data with recommendation requests and enabling instant de-personalization. We will investigate whether the superior offline performance of VMIS-kNN translates to increased user engagement in real-world recommendation systems. To the best of our knowledge, we present the first empirical evidence that the superior predictive performance of VMIS-kNN/VS-kNN observed in offline evaluations holds in a real-world e-commerce setting. Our production system, Serenade, achieves a significant increase in a business-specific engagement metric, outperforming the legacy system by several percent.

3.1 Introduction

Session-based recommendation targets a core scenario in e-commerce and online browsing. Given a sequence of interactions of a visitor with a selection of items, we want to recommend to the user the next item(s) of interest to interact with [63, 67, 70, 93]. This machine learning problem is crucial for e-commerce platforms [51].

This chapter was published as B. Kersbergen, O. Sprangers, and S. Schelter. Serenade – Low-latency session-based recommendation in e-commerce at scale. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 150–159, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392495. doi: 10.1145/3514221.3517901.

Challenges in scaling session-based recommendation. Scaling session-based recommendation systems is a difficult undertaking, because the input space (sequences of item interactions) for the recommendation system is exponentially large (of size $|\mathbf{I}|^n$, where I is the set of all possible items, and n denotes all possible session lengths), making it impractical to precompute recommendations offline and serve them from a data store. This is in stark contrast to classical recommendations based on collaborative filtering [57, 98], which are relatively static as they rely on long-term user behavior [99]. Instead, session-based recommendations have to maintain state in order to react to online changes in the evolving user sessions, and compute next item recommendations with low latency [5, 51] in real-time.

Recent research indicates that nearest-neighbor methods provide state-of-the-art performance for session-based recommendation, and even outperform complex neural network-based approaches in offline evaluations [51, 70]. It is however unclear whether this superior offline performance also translates to increased user engagement in real-world recommendation systems. Furthermore, it is unclear whether the academic nearest-neighbor approaches scale to industrial use cases, where they have to efficiently search through hundreds of millions of historical clicks while adhering to strict service-level-agreements for response latency. This scalability challenge is further complicated by the fact that the applied session similarity functions do not constitute a metric space (e.g., due to the lack of symmetry), which renders common approximate nearest-neighbor search techniques inapplicable.

VMIS-kNN. In order to tackle the scalability challenge, we present *Vector-Multiplication-Indexed-Session-kNN* (VMIS-kNN) in Section 3.4, an adaption of the state-of-the-art session-based recommendation algorithm VS-kNN [70]. VMIS-kNN leverages a pre-built index to compute next-item recommendations in milliseconds for scenarios with hundreds of millions of clicks in historical sessions to search through. Our approach can be viewed as the joint execution of a join between evolving and historical sessions on matching items and two aggregations to compute the similarities. During this joint execution, we minimise intermediate results, control the memory usage and prune the search space with early stopping. As a consequence, VMIS-kNN drastically outperforms VS-kNN in terms of latency and scalability (Section 3.6.2), while still providing the desired prediction quality advantages over neural network-based approaches (Section 3.6.1).

Serenade. Finally, we present the design and implementation of our scalable session-based recommendation system *Serenade*, which employs VMIS-kNN, and can serve a thousand recommendation requests per second with a 90th percentile latency of less than seven milliseconds in scenarios with millions of items to recommend. Our system runs in the Google Cloud-based infrastructure of *Bol*, a large European e-commerce platform, and is in production usage. We discuss design decisions of Serenade, such as stateful recommendation servers, which collocate the evolving user sessions together with update and recommendation requests (Section 3.5.1). Additionally, we describe implementation and deployment details (Section 3.5.2), as well as insights into the remarkably low operational costs of our system (Section 3.7).

Offline and online evaluation. We conduct an extensive evaluation to validate the

predictive performance and low latency of VMIS-kNN in Section 3.6.1. For the Serenade system, we present results from a load test with more than 1,000 requests per second, and the outcome of a three week long online A/B test of our system on the live e-commerce platform in Section 3.6.2. Our system is available under an open license.[1]

Contributions. In summary, in this chapter we contribute the following:

- We present VMIS-kNN, an index-based variant of a state-of-the-art nearest-neighbor algorithm to session-based recommendation, which scales to use cases with hundreds of millions of clicks to search through (Section 3.4).
- We discuss design decisions and implementation details of our production recommendation system Serenade, which applies stateful session-based recommendation with VMIS-kNN, and can handle more than 1,000 requests per second with a response latency of less than seven milliseconds in the 90th percentile (Section 3.5).
- To the best of our knowledge, we provide the first empirical evidence that the superior predictive performance of VMIS-kNN/VS-kNN from offline evaluations translates to superior performance in a real world e-commerce setting; we find Serenade to drastically increase a business-specific engagement metric by several percent, compared to our legacy system (Section 3.6.2).

3.2 Related Work

Research on recommendation systems [15, 16, 28, 36, 38, 73, 83, 94, 107, 124, 129, 131, 132] is a growing field, with a close connection to industry use cases [113, 119, 122], as illustrated by the famous "Netflix Prize" competition [57]. Translating academic progress into deployable solutions has proven to be very difficult [51], as exemplified by the fact that the winning solution of the Netflix prize never went into production [4]. Nearest neighbor-based recommendations, which are in the focus of our work, are a classical approach to recommendation mining [13, 62, 98, 99, 101], and are widely deployed in industry [19–21, 44, 81]. Despite their popularity, these approaches are typically outperformed by matrix factorisation- and deep learning-based methods in offline evaluations on classical collaborative filtering problems [57].

However, recent research indicates that nearest neighbor-based approaches provide state-of-the-art performance and outperform neural networks in sequence-based recommendation tasks. An example for such a task is session-based recommendation, which is in the focus of our work, where recent studies [45, 51, 70] indicate that nearest neighbor-based methods outperform previously proposed neural networks [41, 63, 67, 125]. Similar results have been obtained for the more general sequence-based recommendation task of next basket recommendation (where the set of items in a future shopping basket has to be predicted). Here, the nearest neighbor-based state-of-the-art approach TIFU-kNN [43] and simple popularity-based approaches [64] outperform neural networks as well.

[1] https://github.com/bolcom/serenade

Algorithm 2 Vector-Session-kNN.

1: **function** VS-KNN($\mathbf{s}^{(t)}, \mathbf{H}, \pi, \lambda, m, k$)
2: *Input:* Evolving session $\mathbf{s}^{(t)}$, set of historical sessions $\mathbf{H}$, decay function π,
3: match weight function λ, sample size m, number of neighbors k.
4: *Output:* Scored list of recommended next items $\mathbf{d}$.
5: $\mathbf{H}_s \leftarrow$ historical sessions that share at least one item with $\mathbf{s}$
6: $\overline{\mathbf{H}}_s \leftarrow$ recency-based sample of size m from $\mathbf{H}_s$
7: $\mathbf{N}_s \leftarrow k$ closest sessions $\mathbf{h} \in \overline{\mathbf{H}}_s$ according to similarity $\pi(\omega(\mathbf{s}^{(t)}))^\top \mathbf{h}$
8: **for** each item i occuring in the sessions $\mathbf{N}_s$ **do**
9: $d_i \leftarrow \sum_{\mathbf{n} \in \mathbf{N}_s} \mathbb{1}_n(i) \cdot \frac{1}{|\mathbf{s}^{(t)}|} \cdot \lambda(\max(\omega(\mathbf{s}^{(t)}) \odot \mathbf{n})) \cdot \pi(\omega(\mathbf{s}^{(t)}))^\top \mathbf{n} \cdot (1 + \log \frac{|H|}{h_i})$
return item scores $\mathbf{d}$

3.3 Background

We introduce session-based recommendation and the Vector-Session-kNN method. Given an evolving session (a sequence of interactions with a set of items $\mathbf{I}$) at time t, the goal of session-based recommendation is to accurately predict the next item that the user will interact with at time $t+1$.

Vector-Session-kNN. Vector-Session kNN (VS-kNN) [70] is a state-of-the-art nearest neighbor based approach to session-based recommendation, which outperforms current deep learning approaches for this task. In VS-kNN, we have a set of historical sessions $\mathbf{H} \in \{0,1\}^{|\mathbf{I}|}$ represented as binary vectors in item space, and an evolving user session $\mathbf{s}^{(t)} \in \{0,1\}^{|\mathbf{I}|}$ at time t, as well as a function $\omega(\mathbf{s})$ which replaces the non-zero entries of $\mathbf{s}$ with integers denoting the insertion order of the items in $\mathbf{s}^{(t)}$. Algorithm 2 describes how VS-kNN computes its recommendations for an evolving session $\mathbf{s}^{(t)}$. First a recency-based sample $\overline{\mathbf{H}}_s$ of size m is taken from all historical sessions $\mathbf{H}_s$ that share at least one item with the evolving session (Lines 5 and 6). Next, we compute the k closest sessions $\mathbf{N}_s$ from $\overline{\mathbf{H}}_s$ according to the similarity $\pi(\omega(\mathbf{s}^{(t)}))^\top \mathbf{h}$ (Line 7), which applies an element-wise decay function π to the entries denoting the insertion order in the evolving session. All items occurring in these neighboring sessions are finally scored (Lines 8 and 9) by summing their similarities (the previously computed decayed dot product) weighted by a non-linear function λ applied to the position $\max(\omega(\mathbf{s}^{(t)}) \odot \mathbf{n})$ of the most recent shared item between the evolving session $\mathbf{s}^{(t)}$ and the neighbor session $\mathbf{n}$. The session similarity contribution is additionally weighted by a factor of one over the session length, and by a factor of one plus the "inverse document frequency" $\log \frac{|H|}{h_i}$ of the item, where h_i denotes the number of historical sessions containing item i (a common technique from information retrieval to de-emphasise highly frequent items). Note that the indicator function $\mathbb{1}_n(i)$ is one if item i occurs in the historical session $\mathbf{n}$ and zero otherwise.

Toy example. We provide a toy example for the session similarity and match weighting computation executed by VS-kNN. Assume that we have an evolving session $\mathbf{s}^{(t)} =$ `[0 1 1 0 1]` representing interactions with the three items `[1,2,4]` and a historical session $\mathbf{h} =$ `[0 0 1 0 1]` representing interaction with the items `[2,4]`. The function ω gives us the chronological insertion order for the evolving session, e.g., $\omega(\mathbf{s}^{(t)}) =$ `[0 1 2 0 3]`

of the items in $\mathbf{s}^{(t)}$, starting from the first item (item 1 with insertion time 1) to the most recent item (item 4 with insertion time 3). The insertion order is used to weight matches between the items of the evolving session and the historical session, and the weights are determined by the decay function π, which is a hyperparameter of VS-kNN. A common choice for π is to divide the insertion time by the session length, e.g., $\pi(\omega(\mathbf{s}^{(t)})_i) = \omega(\mathbf{s}^{(t)})_i \,/\, ||\mathbf{s}^{(t)}||_1$. The similarity is finally determined by computing the decayed dot product $\pi(\omega(\mathbf{s}^{(t)}))^\top \mathbf{h}$ between the evolving session $\mathbf{s}^{(t)}$ and historical session $\mathbf{h}$ as the sum of the decayed weights for the intersection of the sessions (the shared items), e.g., $\pi(\omega(\mathbf{s}^{(t)}))^\top \mathbf{h} = [0\ \frac{1}{3}\ \frac{2}{3}\ 0\ \frac{3}{3}]^\top\ [0\ 0\ 1\ 0\ 1] = \frac{2}{3} + \frac{3}{3} = \frac{5}{3}$.

After finishing the session similarity computation, VS-kNN computes item scores from the similarities (Lines 8 and 9). The score for an item is the weighted sum of similarities with $\mathbf{s}^{(t)}$ from the k closest historical sessions $\mathbf{n} \in \mathbf{N}_s$ in which the item occurs. The weights for this sum are computed by the matching function λ, which is applied to the insertion time $\max(\omega(\mathbf{s}^{(t)}) \odot \mathbf{n})$ of the most recent shared item between $\mathbf{s}^{(t)}$ and $\mathbf{n}$. The default choice for λ in VS-kNN is $1 - (0.1 \cdot (\max(\omega(\mathbf{s}^{(t)}) \odot \mathbf{n})))$ for insertion times less than 10 and zero otherwise. For our toy example, the contribution of the matching function for $\mathbf{h}$ looks as follows: $\lambda(\max(\omega(\mathbf{s}^{(t)}) \odot \mathbf{h})) = \lambda(\max(\,[0\ 1\ 2\ 0\ 3]\ \odot\ [0\ 0\ 1\ 0\ 1]\,)) = \lambda(\max(\,[0\ 0\ 2\ 0\ 3]\,)) = \lambda(3) = 0.7$.

3.4 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* (VMIS-kNN).

VMIS-kNN operates on an index structure $(\mathbf{M}, \mathbf{t})$, which we build from a large dataset of historical sessions. We create a hash index $\mathbf{M}$ from an item i to an array $\mathbf{m}_i$ 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 $\mathbf{m}_i$ 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 $\mathbf{m}_i$). The key benefit of this data structure is to allow us amortised constant-time access to the m most recent sessions containing an item.

Furthermore, we maintain an array $\mathbf{t}$ where an entry t_h denotes the integer timestamp for a historical session h. This again provides constant time random access during the online computation of the session similarity score across all the items in an evolving session, as we use consecutive integer identifiers for historical sessions. Algorithm 3 describes the individual steps and data structures that VMIS-kNN leverages for efficient session-based recommendation based on our index data structure.

Index-based session similarity computation. At the heart of VMIS-kNN is the efficient computation of the neighbor sessions $\mathbf{N}_s$ for an evolving session $\mathbf{s}^{(t)}$ using our previously introduced index structure $(\mathbf{M}, \mathbf{t})$ in the function $\mathrm{neighbor\_sessions\_from\_index}$ in Line 8.

We first initialize a set of temporary hashmaps and heaps (Line 11) which serve as

buffers for intermediate results during the computation. Next, VMIS-kNN starts the *item intersection loop*, which iterates over the items in an evolving session $\mathbf{s}^{(t)}$ in reverse order (Line 12). Our approach processes an evolving session $\mathbf{s}^{(t)}$ in inverse insertion order, such that the most recent (and therefore most important) items of an evolving session are visited first. We then add the item identifier i to the temporary hashset $\mathbf{d}$, such that duplicate items in the evolving session can be skipped (Lines 13–14). Next, we look up the item in our inverted index $\mathbf{M}$ to obtain the vector $\mathbf{m}_i$ containing up to m historical session identifiers (Line 15). We then compute the decay score π_i based on the item's position in the evolving session (Line 16).

Now, we start a loop over each historical session j in $\mathbf{m}_i$ (Line 17). If we have already encountered this historical session for a different item, we add the current decay score π_i to the session score r_j (Line 18). However, if the historical session is not yet part of our temporary similarity score hashmap $\mathbf{r}$, we first obtain the timestamp t_j of the historical session (Line 20). If our temporary similarity score hashmap $\mathbf{r}$ contains less than m items, we insert the session identifier j and session similarity score r_j as (key, value)-pair into $\mathbf{r}$, and we insert the session identifier j and session timestamp t_j as (key, value)-pair into a min-heap $\mathbf{b}_t$ (Lines 21–24). If our temporary similarity score hashmap $\mathbf{r}$ already contains m sessions, we need to investigate whether to remove the oldest session. Therefore, we first retrieve the oldest session and corresponding timestamp from the heap $\mathbf{b}_t$ (Line 26).

If the current historical session j is more recent than the oldest session, we need to remove the oldest session from our temporary similarity score hashmap $\mathbf{r}$ and heap $\mathbf{b}_t$, and update both with the values from the current historical session j (Lines 27–31). Finally, we extract the top-k scored sessions from the max-heap $\mathbf{N}_s$ in the *top-k similarity loop* and return them (Line 33).

VMIS-kNN computes the final item scores by using the pre-computed session similarity r_n for a neighboring historical session $\mathbf{n}$. We however simplify the item scoring function from Line 9 of Algorithm 2 in two ways: (i) we remove the constant factor $1/|\mathbf{s}^{(t)}|$ applied to each similarity (which does not change the neighbor ranking); and (ii) we use a weight of $\log \frac{|H|}{h_i}$ instead of $(1 + \log \frac{|H|}{h_i})$ for the similarities, which gives us better results in offline evaluations on held-out data.

A particular advantage of VMIS-kNN is its support for early stopping, which allows us to skip certain historical sessions during the similarity computation: we can immediately break the session for-loop if our current historical session j is older than the eldest session l in our heap $\mathbf{b}_t$ as $\mathbf{m}_i$ is already sorted in descending timestamp order, and will not contain more recent sessions in later positions (Line 32).

Time complexity. The time complexity of the online computation of our similarity score is dominated by the linear time required to execute the three for-loops (Lines 12, 17 and 33) and the logarithmic time required to modify the heaps $\mathbf{b}_t$ (Lines 24, 31) and $\mathbf{N}_s$ (Lines 34, 37, 38), yielding a theoretical time complexity of $O(|\mathbf{s}^{(t)}| \cdot m \cdot \log_2 m + m \cdot \log_2 k) = O(|\mathbf{s}^{(t)}| \cdot m \cdot \log_2 m)$ as $k \leq m$. Thus, the time complexity only depends on: (i) the number of items in the evolving session $\mathbf{s}^{(t)}$, which we cap at a maximum value, and (ii) the number m denoting how many recent historical sessions to consider. Hence, the time complexity of our implementation is (theoretically) independent of the number of historical sessions $|\mathbf{H}|$ and the number of unique items $|\mathbf{I}|$ in our dataset. As

Algorithm 3 Vector-Multiplication-Indexed-Session-kNN (VMIS-kNN)

function VMIS-KNN($\mathbf{s}^{(t)}$, $(\mathbf{M}, \mathbf{t})$, π, λ, m, k)
  Input: Evolving session $\mathbf{s}^{(t)}$, session similarity index $(\mathbf{M}, \mathbf{t})$, decay function π,
    sample size m, match weight function λ, number of neighbors k.
  Output: Scored list of recommended next items $\mathbf{d}$.
  $(\mathbf{N}_s, \mathbf{r}) \leftarrow$ neighbor_sessions_from_index($\mathbf{s}^{(t)}$, $(\mathbf{M}, \mathbf{t})$, π, m, k)
  for each item i occurring in the sessions $\mathbf{N}_s$ **do**
    $d_i \leftarrow \sum_{\mathbf{n} \in \mathbf{N}_s} \mathbb{1}_n(i) \cdot \lambda(\max(\omega(\mathbf{s}^{(t)}) \odot \mathbf{n})) \cdot r_n \cdot \log \frac{|H|}{h_i}$
  return item scores $\mathbf{d}$

function NEIGHBOR_SESSIONS_FROM_INDEX($\mathbf{s}^{(t)}$, $(\mathbf{M}, \mathbf{t})$, π, m, k)
  initialize hashmap $\mathbf{r}$ for temporary similarity scores, min-heap $\mathbf{b}_t$ of capacity m for the most recent similar historical sessions, hashset $\mathbf{d}$ for already processed items, max-heap $\mathbf{N}_s$ of capacity k for closest sessions
  for item $i \in \mathbf{s}^{(t)}$ in reverse insertion order **do** ▷ *Item intersection loop*
    if $i \notin \mathbf{d}$ **then**
      insert i into $\mathbf{d}$
      $\mathbf{m}_i \leftarrow$ most recent sessions for item i from inverted index $\mathbf{M}$
      $\pi_i \leftarrow$ decay weight $\pi(\omega(\mathbf{s}^{(t)}))_i$ of item i in session $\mathbf{s}^{(t)}$
      for session $j \in \mathbf{m}_i$ **do**
        if $j \in$ keys($\mathbf{r}$) **then** $r_j \leftarrow r_j + \pi_i$
        else
          $t_j \leftarrow$ timestamp of session j fetched from index $\mathbf{t}$
          if $|\mathbf{r}| < m$ **then**
            $r_j \leftarrow \pi_i$
            insert (j, r_j) into $\mathbf{r}$
            insert (j, t_j) into $\mathbf{b}_t$
          else
            $(l, t_l) \leftarrow$ current heap root of $\mathbf{b}_t$
            if $t_j > t_l$ **then**
              $r_j \leftarrow \pi_i$
              remove (l, r_l) from $\mathbf{r}$
              insert (j, r_j) into $\mathbf{r}$
              update heap root of $\mathbf{b}_t$ with (j, t_j)
              else break

  for $(j, r_j) \in \mathbf{r}$ **do** ▷ *Top-k similarity loop*
    if $|\mathbf{N}_s| < k$ **then insert** (j, r_j) into $\mathbf{N}_s$
    else
      $(n, r_n) \leftarrow$ current heap root of $\mathbf{N}_s$
      if $r_j > r_n$ **then update** heap root of $\mathbf{N}_s$ with (j, r_j)
      else if $r_j = r_n$ **and** $t_j > t_n$ **update** heap root of $\mathbf{N}_s$ with (j, r_j)
  return $\mathbf{N}_s$

a micro-optimisation, we leverage octonary heaps [18] instead of binary heaps, which have more children per node, and therefore provide better performance for workloads like ours with frequent insertions.

Space complexity. The space complexity of the index for VMIS-kNN is dominated by the storage cost $O(|\mathbf{I}| \cdot m)$ for the hashmap holding the inverted index $\mathbf{M}$, which maps unique item indices to the m most recent historical sessions containing the item.

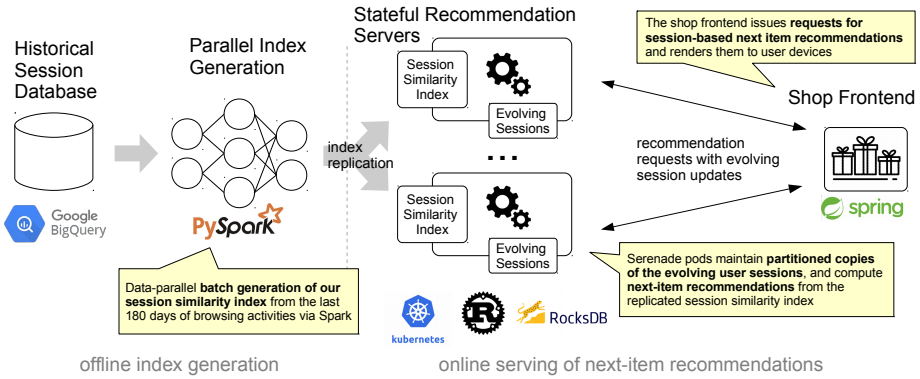


Figure 3.1: High level architecture of the Serenade recommendation system. The offline component (left) generates a session similarity index ❶ from several billion historical click events via a parallel Spark job in regular intervals. The online serving machines (right) maintain state about the evolving 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.

From a classical query processing perspective, VMIS-kNN conducts two aggregations (identifying the m most recent sessions with an item match, and computing their similarities) on the result of a join between the items of the evolving session $\mathbf{s}^{(t)}$ and the historical sessions $\mathbf{H}$. The efficiency of VMIS-kNN derives from the fact that we jointly execute the join and subsequent aggregations in Algorithm 3, while only maintaining intermediate results of a size proportional to the final outputs (instead of first materialising the potentially large complete join result before running the aggregations).

3.5 Serenade

We present the design and implementation of our scalable recommendation system *Serenade*, which leverages VMIS-kNN (Section 3.4) and provides recommendations on the product detail pages of Bol.

3.5.1 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 offline 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 recommendations 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 3.1, 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 collocate 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 [23].

3.5.2 Implementation

The high-level architecture of Serenade (derived from our design decisions in Section 3.5.1) is illustrated in Figure 3.1. Serenade consists of two components: The offline component (shown in the left part of the figure) builds the session index from click data and is implemented as an Apache Spark pipeline. The online component (shown in the right part of the figure) computes and serves session-based recommendations with VMIS-kNN, and is implemented as a REST application. Note that Serenade builds upon existing Google Cloud infrastructure rented by Bol.

Offline index generation. The index generation ❶ from historical click data is implemented as a parallel dataflow computation in Apache Spark using Spark MLLib pipeline steps [77] as abstraction, and executed in regular intervals (typically once per day) in Google Dataproc. It uses historical click data from the last 180 days of our platform (stored in Google BigQuery) as its input, which amounts to roughly 2.3 billion user-item interactions. The output of the Spark job is a compressed representation of our index, stored in the distributed filesystem in the Apache Avro format. The index data is later on ingested by Serenade's serving component, where it requires around 13 gigabytes of memory.

Online serving of next item recommendations. The serving component of Serenade is responsible for computing next item recommendations with VMIS-kNN in response to session updates. We implement this serving component in Rust, as a web application based on the Actix [2] framework. The shopping frontend contacts our Serenade servers whenever a user generates new item interactions in their session (e.g., by visiting a product detail page). The Serenade servers update the state of the evolving user session ❷, and respond to the shopping frontend with a list of 21 recommended next items for the user (the number of items required by the UI in the frontend) based on a VMIS-kNN prediction ❸. The VMIS-kNN predictions leverage the previously computed offline index. We additionally apply business rules to the recommendations to remove unavailable products and to filter for adult products.

Colocation of evolving sessions and session updates. As discussed in Section 3.5.1, we need to colocate the evolving sessions with the recommendation requests and session updates to be able to compute up-to-date recommendations with low latency. We maintain the evolving sessions in a local key-value store (RocksDB [96]) directly on the serving machines, to avoid additional network reads and writes. For collocation, we have to partition both the evolving sessions and the recommendation requests (which also contain the session updates) over the serving machines, based on their session identifier. In order to guarantee that all the update/recommendation requests for a particular session are always handled by the same machine, we configure request routing via "sticky sessions" provided by Kubernetes' session affinity functionality [59]. The communication with RocksDB turns out to be extremely fast; in a microbenchmark with 10 million operations for our workload, we found the 99th percentile of the read latency to be 5 microseconds, and the 99th percentile of the write latency to be 18 microseconds. This colocation approach provides a big latency improvement over network reads and writes to a distributed key-value store like BigTable, where the response latency for lookups is already 15ms on the 99.5 percentile in our experience.

Discussion. Our colocation approach can be viewed as a trade-off between reducing the response latency and guaranteeing fault tolerance for the session data, as the session data could be temporarily lost in cases of machines failures or elastic scaling of the machine pool. However, this turns out to be no problem in practice for several reasons: (i) Our service proved to be very stable, we encountered no issues in a long A/B test running for several weeks (details will be described in Section 3.6.2), where no elastic scaling was required, as a small set of cheap machines with a low number of cores could reliably handle hundreds of request per second; (ii) The sessions are very short-lived anyways, we only leverage the most recent interactions for recommendations (which also have the highest impact on the session similarities), their loss would not have a drastic impact, as the recommendation would quickly collect new interactions; and (iii) The sessions are additionally tracked by other parts of our e-commerce platform for analytics. It is not the task of the recommendation system to store them permanently, on the contrary, we configure RocksDB to remove the data for a session after 30 minutes of inactivity.

Deployment. We deploy our recommendation servers via a Docker image managed by Kubernetes. The image is created by our continuous integration infrastructure, and we leverage a multi-stage build. In the first stage, we download all dependencies and compile our Rust application (which results in a large image with a size of several gigabytes); in the second stage, we reduce the size of this image by only retaining the compiled application and the runtime dependencies. The image for Serenade is then pushed into a Docker repository. The application is deployed to a Google Kubernetes Engine cluster, alongside with load balancing pods (`istio sidecars` [106]) which provide us with the session affinity routing required for colocating the evolving user sessions and recommendation requests on our machines.

Depersonalisation. We are required to provide non-personalised recommendations for users who do not give consent to leverage their session history for personalisation. This is comparatively easy to implement with VMIS-kNN: we create a non-personalised

Table 3.1: Public and proprietary datasets for evaluation.

	retailr	*rsc15*	*ecom-1m*	*ecom-60m*	*ecom-90m*	*ecom-180m*
clicks	86,635	31,708,461	1,152,438	67,017,367	89,883,761	189,317,506
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

variant which only leverages the currently displayed item on the product detail page for recommendation. This depersonalisation can be applied in real-time (e.g., when a user revokes their consent to personalisation), as each request from the shop frontend includes a binary flag denoting the status of the user consent.

3.6 Experimental Evaluation

In the following, we first evaluate the prediction quality and index design of VMIS-kNN in Section 3.6.1, and subsequently evaluate the scalability and business performance of Serenade in offline experiments and an online A/B test (Section 3.6.2). We provide the code for our experiments.[2]

Datasets. We leverage a combination of public and proprietary click datasets from e-commerce for our offline experiments. We experiment with the publicly available [95] datasets *retailrocket* (an e-commerce dataset from the company "Retail Rocket") and *rsc15* (a dataset used in the 2015 ACM RecSys Challenge), which are commonly used in comparative studies on session-based recommendation [70]. In addition, we create the non-public datasets *ecom-1m*, *ecom-60m*, *ecom-90m* and *ecom-180m* by sampling data from our e-commerce platform with increasing numbers of clicks. The statistics of these datasets are shown in Table 3.1. Each dataset consists of tuples denoting the `session_id`, `item_id` and `timestamp` of a click event on the platform.

Our proprietary dataset *ecom-180m* is more than six times larger than the largest publicly available dataset *rsc15*. We additionally show statistics of the distribution of clicks per session in the form of its 25th, 50th, 75th and 99th percentile. We find that the majority of sessions on e-commerce platforms is very short (e.g., the median number of clicks per session is less than five) and that these statistics are very similar across all six datasets. In the tail, the sessions from our platform are about twice as long though compared to the public datasets (e.g., the 99th percentile is around 38 clicks in our data and 19 clicks in the public datasets).

[2] https://github.com/bolcom/serenade-experiments-sigmod

3.6.1 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 [51, 70]).

Experimental setup. We replicate the setup from [51, 70], and compare the predictive performance of VMIS-kNN against three recent neural network-based approaches to session-based recommendation (GRU4Rec [41], NARM [63] and STAMP [67]) 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.

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*.

Results and discussion. We first investigate the Mean Average Precision (MAP@20), Precision (Prec@20) and Recall (R@20), which denote to what extent an approach correctly predicts the next items in a session. VMIS-kNN outperforms the neural approaches in all of these metrics. Its MAP@20 is .0268 compared to .0251 for the best performing neural approach (GRU4Rec); the Prec@20 of VMIS-kNN is .0722 compared to .0680 for the best performing neural approach (NARM in this case); and VMIS-kNN's R@20 is .378 compared to .359 for the best performing neural approach GRU4Rec. We additionally look at the Mean Reciprocal Rank (MRR@20), which puts a stronger weight on the immediate next item in a session. Again, VMIS-kNN outperforms all neural-based approaches with an MRR@20 of .286 compared to .255 for the best performing neural method (GRU4Rec in this case).

In summary, we confirm that the findings from recent studies on the state-of-the-art performance of VS-kNN also hold for VMIS-kNN on our proprietary data. It is an open question, why neural networks do not outperform conceptually simpler methods in sequential recommendation. There is recent evidence that neural networks have difficulties capturing item frequency information [43], and that many researchers do not adequately compare their proposed neural methods against simple baselines [64, 70].

Sensitivity to Hyperparameter Choices

Next, we investigate the sensitivity of VMIS-kNN to its hyperparameters: the number of neighbors k and the number of most recent sessions per item m.

Experimental setup. We run an exhaustive grid search over 55 combinations of the hyperparameters (the k most similar sessions out of the m most recent sessions) for our four large datasets *ecom-60m*, *ecom-90m*, *ecom-180m* and *rsc15*, where we use the last day as held-out test set.

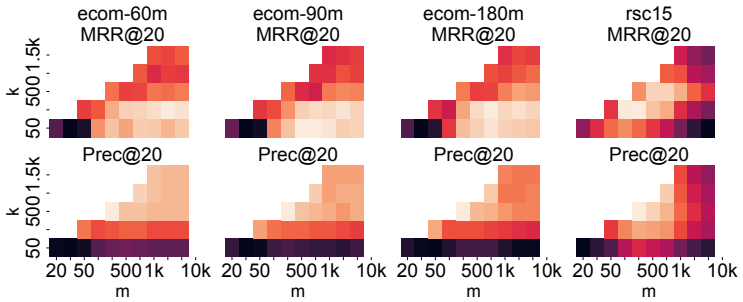


Figure 3.2: Sensitivity of MRR@20 and Prec@20 to the hyperparameters k (the number of neighbors) and m (the number of most recent sessions per item) in our proprietary datasets.

Results and discussion. Figure 3.2 illustrates the results of the grid search for MRR@20 and Prec@20 on our datasets with a heatmap where lighter colors indicate better metric values. We observe a unimodal distribution of the resulting metric values for each dataset and metric. The results differ (i) based on dataset, e.g., all samples from our proprietary data show similar outcomes, while the distribution for *rsc15* is very different; and (ii) based on metric, e.g., hyperparameters that work well for MRR (which focuses on the position of the first correctly predicted relevant item) do not necessary provide the best performance for Precision (which considers all correctly predicted relevant items). Our results indicate that VMIS-kNN is easy to tune via offline grid search for a given dataset and target metric.

Index Design

Next, we run a microbenchmark comparing VMIS-kNN vs. VS-kNN to validate the performance of our index-based similarity computation.

Experimental setup. We experiment with our index and similarity computation (refered to as *VMIS-kNN*) from Section 3.4 and compare it against two baseline implementations: (i) *VS-kNN* – a baseline implementation that mimics VS-kNN's similarity computation by holding the historical data in hashmaps, and first identifying the m most recent sessions with at least one shared item before computing the similarities, and (ii) *VMIS-kNN-no-opt*, a basic variant of VMIS-kNN, which does not contain several optimisations such as early stopping or using octonary heaps instead of binary heaps.

We conduct a micro-benchmark on the *ecom-1m* dataset. We ask each variant to compute the k closest sessions for the sessions from the test set, and we randomly pick the number of items (e.g., the session length) for each session to include in the computation. We repeat this experiment ten times for various values of m (the number of most recent sessions to consider) with six threads, and measure the execution times for $k = 100$ (trying other values of k did not significantly change the results). We implement all algorithms in Rust 1.54 and run the comparison on a machine with an i9-10900KF CPU @ 3.7GHz with ten cores and 64GB of RAM, running Windows 10 21H1.

Results and discussion. The bottom plot in Figure 3.3 shows the resulting runtimes

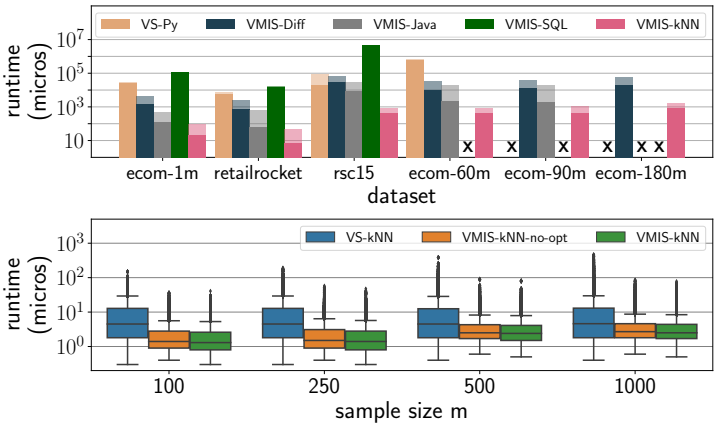


Figure 3.3: (Top) Median and 90th percentile of the computation time per session in microseconds (log-scale) for different VMIS-kNN implementations; (Bottom) Microbenchmark runtimes in microseconds (log-scale) for VMIS-kNN vs. VS-kNN on the *ecom-1m* dataset with k = 100.

in microseconds for each of our variants. The results are consistent across all values of m: We find that both *VMIS-kNN* and *VMIS-kNN-no-opt* drastically outperform the *VS-kNN* baseline by a factor of three to five. We attribute this observation to the optimised access patterns in the index of VMIS-kNN, which allows us to avoid costly set intersection operations, and the minimisation of intermediate results with our heap data structures (Section 3.4). We furthermore observe that *VMIS-kNN* consistently outperforms *VMIS-kNN-no-opt* by 6% to 12% which validates our micro optimisations such as early stopping and leveraging octonary heaps instead of binary heaps.

3.6.2 Serenade

Next, we evaluate our Serenade system. We validate our implementation choices, run a load test for the system, and finally present the results from a three week long A/B test on the live platform.

Validation of Implementation Choices

We present an offline experiment which focuses on the performance of our index-based VMIS-kNN approach. We compare our Rust-based implementation against implementations in other programming languages and computational engines to validate our design choice of a custom implementation in Rust. Note that we provide the source code for the alternative implementations in our experiment repository as well.

Experimental setup. We compare our Rust-based VMIS-kNN implementation against four other implementations:

- **VS-Py** – a Python-based implementation of the original VS-kNN approach, based

on the reference code [116] from the original VS-kNN paper; we expect this variant to be non-competitive as it is a mere research implementation;

- **VMIS-Diff** – an implementation of VMIS-kNN in *Differential Dataflow* [75], which computes the recommendations incrementally via joins and aggregations; this variant allows us to evaluate the benefits of an incremental similarity computation for growing sessions;
- **VMIS-Java** – an implementation of VMIS-kNN in Java, which stores the historical session data in Java hashmaps; the purpose of this variant is to evaluate the effects of not having full control over the memory management during the similarity computation (and instead relying on a garbage collector);
- **VMIS-SQL** – an implementation of VMIS-kNN in SQL, which leverages the embeddable analytical database engine *DuckDB* [88] in version 0.2.2; the purpose of this variant is to evaluate whether a custom implementation of the approach is necessary; we note that we found it very difficult to express the similarity computation in plain SQL, as it required several deeply nested subqueries;

We ensure through evaluations on held-out data that all variants are correctly implemented and provide equal predictive performance. We expose the historical session data from our public and proprietary datasets to each of these baselines. Next, we ask each implementation to sequentially compute next-item recommendations with a single thread for the growing evolving sessions in the test set of each dataset, and measure the prediction time in microseconds. We run each implementation on a `n1-highmem-8` instance in the Google cloud with 50 gigabytes of RAM, and use $m = 5000$ and $k = 100$ as hyperparameter settings.

Results and discussion. The top plot of Figure 3.3 illustrates the resulting runtimes from our experiment for the different datasets and baseline implementations. Note that we plot the median and 90th percentile (p90) of these runtimes on a logarithmic scale in a single bar, where the lighter top part denotes the 90th percentile runtime. Our VMIS-kNN implementation consistently outperforms all the baselines both in terms of median and p90 runtime; it is more than two orders of magnitude faster than the Python reference implementation, and more than one order of magnitude faster than the differential dataflow implementation.

The second-best implementation is the Java baseline, which is still outperformed by an order of magnitude for the 90th percentile runtime on all datasets except the small *ecom-1m* dataset. When we look at the results for larger datasets, we additionally observe that several baselines start to encounter memory issues (even though they can use 50 gigabytes of RAM), and fail to complete the computation. This happens for the Python implementation (which relies on pandas dataframes internally), for the SQL implementation as well as for the Java variant.

Note that our Serenade implementation provides a p90 runtime of at most 1.7 milliseconds on all datasets. We attribute this to the fact that our implementation allows us to carefully control memory allocation and to avoid the materialisation of large intermediate results (such as the complete set of item matches with the historical sessions). We observe that the differential implementation always manages to compute

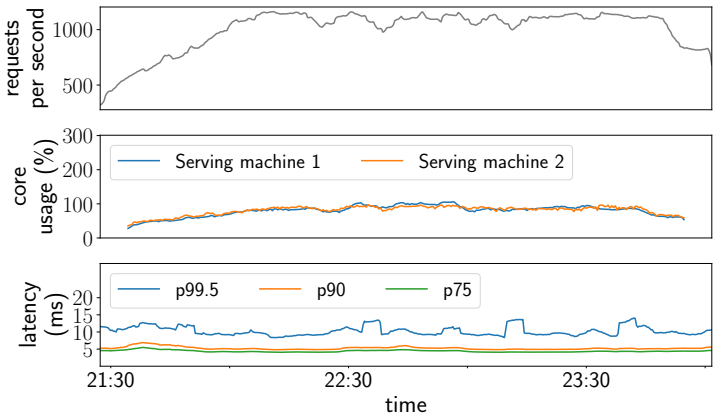


Figure 3.4: Requests per second, core usage in percent and response latency during a load test with more than 1,000 requests per second. Serenade handles about 500 requests per second per core with a 90th percentile latency of less than seven milliseconds.

results; however, the incremental computation does not pay off runtime-wise, because differential dataflow has to index all intermediate results due to its support for updates in response to input data changes (which is not required in our use case). Finally, we find the SQL implementation to be non-competitive and to not scale to large datasets, which we attribute to the large intermediates from the nested subqueries, and which confirms that a custom implementation of VMIS-kNN is more suitable to scale to large datasets.

Offline Load Test

We finally run a load test in our staging environment to validate that Serenade is able to handle peak production workloads.

Experimental setup. We leverage a setup that resembles our production environment: Serenade's index is built from the last 180 days of browsing activities, covering 6.5 million distinct items. We deploy Serenade on two Kubernetes pods, running on shared core `n1-standard-16` instances in the Google Cloud, where each pod gets provisioned with three cores from an Intel Xeon CPU @ 2.00GHz and 16 GB of RAM.

We generate a simulated load of more than 1,000 requests per second by replaying historical traffic via a load generator application for several hours. We measure the response latency of Serenade as well as the core usage on the machines.

Results and discussion. Figure 3.4 plots the resulting response latency and core usage for our load test. We find that Serenade gracefully handles the load of more than 1,000 requests per second, and responds within less than 7 milliseconds in 90 percent of the cases (`p90`) and in less than 15 milliseconds in 99.5% of the cases (`p99.5`). Each instance uses only one of the three provisioned cores for most of the time. We base our production experiments in the following on the outcomes of this load test.

Online Evaluation in an A/B Test

We present results from a three week long online A/B test on our e-commerce platform, where we compared two variants of Serenade against our existing legacy recommendation system (referred to as *legacy*), which applies a variant of classic item-to-item collaborative filtering [98].

Experimental setup. We show Serenade's recommendations on the product detail page of our e-commerce platform, in a slot titled 'other customers also viewed'. We evaluate two different variants of Serenade: the first variant *serenade-hist* leverages the last two items from each evolving session to compute predictions, while the second variant *serenade-recent* only leverages the most recent item. We set the hyperparameters of VS-kNN to $m = 500$ and $k = 500$, which provide a reasonable trade-off between prediction quality in offline experiments and index size. We run the test for 21 days, in which more than 45 million randomly assigned user sessions were subjected to the recommendations from our variants. We ensure that both the legacy system and Serenade consume the same click data as input at the same time (once per night). Serenade builds its index from the last 180 days of data; after filtering, its daily training data consists of around 111 million sessions with 582 million distinct user-item interactions and contains 6.5M distinct items. We measure the request load to the recommendation system, the response latency (as experienced from the shop frontend) and several business-specific engagement metrics.

Results and discussion. We discuss the systems- and business-specific outcomes of our A/B test.

Response latency. Our most important systems-related metric is the response latency. Our recommendation systems have to adhere to a strict SLA of responding in less than 50 milliseconds, otherwise requests would be discarded. Recent research also indicates that fast response times help with the acceptance of recommendations in general [51]. Our system architecture and implementation decisions (Section 3.5) are tailored to allow for low latency responses of our system. This is confirmed by the experimental results illustrated in Figure 3.5, which plots different percentiles of the response latency distribution over the three weeks of our A/B test, and shows the load of the system (in terms of the number of requests per second) for comparison. The request load varies between 200 and 600 requests per second over the day. We find that Serenade's response latencies are very low, the 90th percentile is consistenly around 5 milliseconds, and even the 99.5th percentile is below 10 milliseconds in the majority of cases. This confirms that Serenade exhibits a consistently fast, and stable low-latency response behavior.

CPU usage and operational cost. We deploy Serenade analogously to the setup from the load test in Section 3.6.2: We leverage two Kubernetes pods, running on shared core `n1-standard-16` instances in the Google Cloud, where each pod gets provisioned with cores of an Intel Xeon CPU @ 2.00GHz and 16 GB of RAM. Even with such low resources, Serenade is able to gracefully handle the request workload. We reconfirm the findings from our load test (Section 3.6.2), as Serenade only exhibits a core usage of less than 36% (less than one core) in cases with over 500 requests per second. We also observe a well-behaved linear scaling (with a gentle slope) of the core usage with the

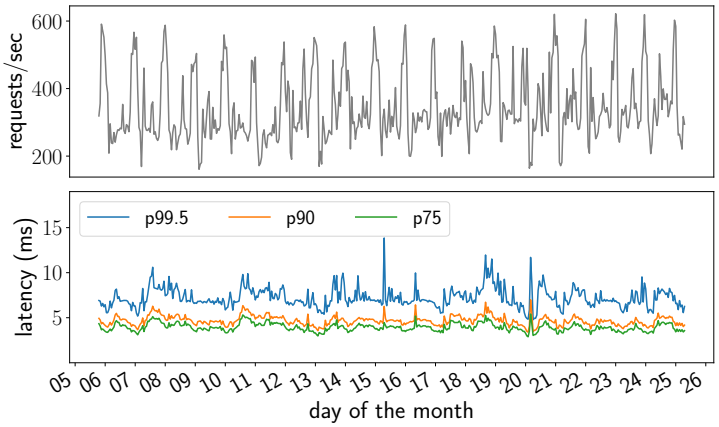


Figure 3.5: Requests per second and response latency per hour during our three week long A/B test on the live platform. Serenade responds within less than seven milliseconds in 90% of the cases, even for peak times with more than 600 requests per second.

number of requests per second.

Customer engagement. Systems-related metrics are important for successfully operating a recommendation system, however in the end the recommendation system has to perform well in business-related metrics to be valuable for an e-commerce platform. As VMIS-kNN outperforms other approaches in offline evaluations (Section 3.6.1), we are interested to determine how this behavior translates to customer engagement in our A/B test. For that, we measure a conversion-related business metric for the engagement with recommendations on the product detail page.

We find that our session-based recommendations drastically increase this engagement metric for the slot on the product detail page. *Serenade-hist* exhibits a 2.85% increase in the business metric (compared to *legacy*), and *serenade-recent* even shows an increase of 5.72% (both findings are statistically significant). When we control for the overall impact on a site-wide level however, we find that *serenade-recent* exhibits a cannibalising behavior, as it drives down the engagement of other slots on the product detail page (e.g., the 'often bought together' slot). We do not observe this effect for *serenade-hist* though, rendering it the preferred variant.

Summary. We find that Serenade easily handles the load of up to 600 requests per second during our A/B test and consistently generates its recommendations with very low response latency (less than seven milliseconds in the 90th percentile). We furthermore find that the session-based recommendations produced by VMIS-kNN significantly increase customer engagement compared to classical item-to-item recommendations (as produced by our *legacy* system). We would like to highlight that, to the best of our knowledge, we are the first to provide empirical evidence that the superior offline performance of VS-kNN/VMIS-kNN also translates to superior performance in terms of business metrics in a live, real recommendation system. This is often not the case for academic recommendation approaches, the winning solution of the highly popularised

Netflix prize, for example, never went into production [4].

3.7 Learnings & Conclusion

In this chapter we presented our nearest neighbor approach VMIS-kNN as well as the design and implementation of our scalable session-based recommendation system *Serenade*. We conducted an extensive offline evaluation of VMIS-kNN and Serenade to validate our design decisions, and detailed results on the latency, throughput, and predictive performance of our recommendation system from an online A/B test with up to 600 requests per second for 6.5 million distinct items on more than 45 million user sessions on Bol's e-commerce platform.

In addition to the contributions listed in Section 3.1, we would like to highlight Serenade's low operational cost: We run two instances with three cores each in the Google cloud (provisioned on shared core `n1-standard-16` instances) for the serving pods, and require 40 minutes on 75 machines of type `n1-highmem-8` for creating the index with Spark every day, which results in a total operational cost of less than 30 euros per day for Serenade. As discussed in Section 3.6.2, Serenade only leverages one of the three cores on each instance, and we only provision the other cores to be prepared for peak loads, e.g., during denial-of-service attacks.

This low cost becomes especially attractive when we compare it with the high cost to train deep learning models. As an example, a neural learning-to-rank model on our platform incurs at least an order of magnitude more costs to be operated on a daily basis, and additionally requires GPU machines for training, which are often a contested resource in the cloud.

In future work, we intend to explore whether we can run our similarity computations on a compressed version of the index, and whether we can incrementally maintain the index with a system such as Differential Dataflow [75].

In the next chapter, we extend our exploration to address the practical challenges of evaluating the inference performance of SBR models at scale, focusing on identifying high-performing and cost-efficient solutions for diverse e-commerce deployment configurations.

4

Etude – Evaluating the Inference Latency of Session-Based Recommendation Models at Scale

In this chapter, we transition from our work on scaling the Session-Based Recommendation (SBR) algorithm VMIS-kNN, which is successfully used in production, to the exploration of our benchmarking framework ETUDE. While VMIS-kNN addressed scalable production-ready recommendations, ETUDE tackles a broader, equally critical challenge in SBR: the efficiency evaluation and deployment of diverse SBR models across different e-commerce use-cases.

Our next research question focuses on:

RQ3 How can we automatically evaluate the inference performance of SBR models under different deployment options?

We answer this question by investigating state-of-the-art neural methods for session-based recommendations from different neural architectures. We design and implement ETUDE, an end-to-end benchmarking framework that enables data scientists to automatically evaluate the inference performance of SBR models under declaratively specified deployment options. ETUDE facilitates this evaluation by allowing users to specify workload statistics, hardware configurations, as well as latency and throughput constraints. Based on these inputs, the framework automatically deploys and executes inference benchmarks in Kubernetes using a synthetically generated click workload. ETUDE then provides detailed throughput and latency measurements, serving as a foundation to identify feasible and cost-effective deployment strategies. We demonstrate the capabilities of ETUDE through an experimental study involving ten SBR models under realistic and challenging settings inspired by real-world workloads at the large European e-commerce platform Bol.

The study in this chapter evaluates performance in various e-commerce use cases for the different brands within Ahold Delhaize,[1] including grocery shopping and broader

This chapter was published as B. Kersbergen, O. Sprangers, F. Kootte, S. Guha, M. de Rijke, and S. Schelter. Etude – Evaluating the inference latency of session-based recommendation models at scale. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 5177–5183, Los Alamitos, CA, USA, May 2024. IEEE Computer Society. doi: 10.1109/ICDE60146.2024.00389.

[1] https://www.aholddelhaize.com/

online retail platforms. We identify optimal deployment options in terms of models and cloud instance types, highlighting their trade-offs in performance and cost. Additionally, our analysis uncovers critical performance bottlenecks in existing solutions, such as the open-source TorchServe inference server from the PyTorch ecosystem and implementations of four SBR models in the open-source RecBole library. We publicly release the ETUDE framework along with our experimental results.

4.1 Introduction

Session-based recommendation (SBR) targets a core scenario in e-commerce. Given a sequence of interactions of a visitor with a selection of items, we want to recommend the next item(s) of interest to interact with [25, 42, 49, 63, 67, 70, 93, 108, 109, 120, 123]. This machine learning problem is crucial for large e-commerce platforms which offer millions of items such as Bol [51, 52].

Challenges in deploying session-based recommendation systems. Scaling session-based recommendation systems is a difficult undertaking, because the input space (sequences of item interactions) for the recommendation system is exponentially large, which renders it impractical to precompute recommendations offline and serve them from a data store. Instead, session-based recommendations have to interactively react to changes in the ongoing user sessions, and compute next item recommendations with low latency [5, 51]. At Bol, for example, an SBR model has to handle at least 1,000 requests per second with a 90-th percentile latency of at most 50 milliseconds [51, 52]. Furthermore, deploying an SBR system often involves choosing from a large number of models [25, 42, 49, 63, 67, 93, 108, 109, 120, 123], implemented in academic libraries like RecBole [128], which lack support for model deployment [89] and inference optimisations [90].

For these reasons, it is currently very difficult for companies to deploy SBR models in challenging production scenarios. Data scientists typically have to prototype different deployment options in collaboration with devops teams, in order to manually evaluate a model's inference performance on a per use-case basis. This is often a tedious process lasting several weeks, which eats up the time of hard-to-hire experts. Furthermore, this process may have to be repeated multiple times in retail corporations such as Ahold Delhaize,[2] which have different brands and platforms in different countries, with varying product catalogs and visitor numbers.

Existing benchmarks are not designed for such industry scenarios unfortunately. The Session-Rec [71] benchmark, for example, measures prediction quality only, with Python-based single-threaded evaluation, and can therefore not assess real-world inference performance. Systems-focused benchmarks like MLPerf [91] are designed to evaluate the performance of inference systems, but not to help data scientists with the choice of models and deployment options in an end-to-end cloud setup.

The ETUDE benchmarking framework. In order to address the issues outlined above, we present the ETUDE benchmarking framework (Section 4.2). ETUDE allows

[2] Ahold Delhaize is an international retail group composed of nineteen companies located across the United States, Europe and Indonesia.

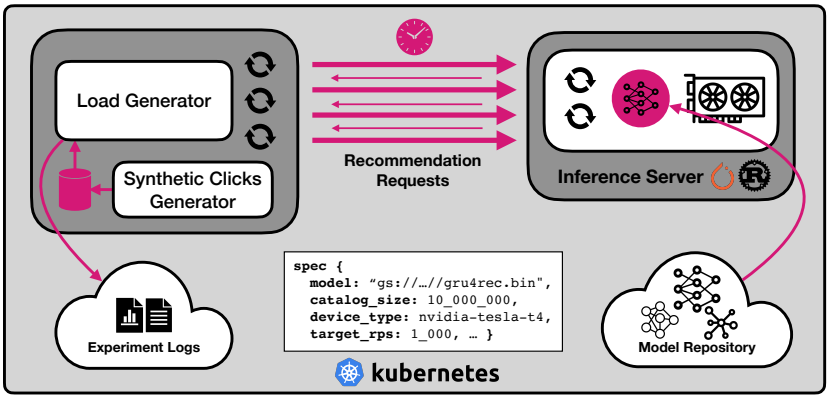


Figure 4.1: High-level overview of ETUDE: Data scientists can automatically evaluate the inference performance of SBR models under declaratively specified deployment options in terms of hardware, workload statistics, as well as latency and throughput constraints.

data scientists to automatically evaluate the inference performance of SBR models under different deployment options. Data scientists provide a set of trained SBR models and declaratively specify statistics of the underlying product catalog, hardware options for deployment (e.g., the type of GPU to use), together with latency and throughput constraints. Based on this, ETUDE automatically deploys and runs an inference benchmark in Kubernetes with a synthetically generated click workload, and provides the data scientists with measurements on throughput and latency, as a basis for deciding on feasible and cost-efficient deployment options.

We discuss the design of ETUDE in Section 4.2, including its synthetic workload generator, its backpressure-aware load generator and our Rust-based inference server. We showcase ETUDE in Section 4.3, where we evaluate ten recently proposed neural SBR models with different deployment options (CPU/GPU inference, small/large product catalogs, just-in-time-optimisation of the underlying models) in challenging settings resembling real-world workloads encountered at Bol.

Contributions. Our contributions in this chapter are as follows:

- We detail the design of our automated end-to-end SBR model benchmarking framework ETUDE (Section 4.2).
- We present an experimental study for ten different SBR models in challenging settings resembling real-world workloads encountered at Bol. We determine performant and cost-efficient deployment options in terms of models and cloud instance types for a variety of online shopping use cases (ranging from grocery shopping to large e-commerce platforms) in Section 4.3.
- We identify severe performance bottlenecks in the open source TorchServe inference server from the PyTorch ecosystem and in the implementations of four SBR models from the open source RecBole library (Section 4.3).

- We make the source code of ETUDE and our experimental results available at https://github.com/bkersbergen/etudelib.

4.2 The Etude Benchmarking Framework

Design goals. We design ETUDE to address the previously outlined challenges. We assume that a data scientist (who is typically an ML expert, but not an expert in systems or devops) wants to assess whether their SBR model can be deployed in production, and what setup (in terms of the number and type of machines) is required to adhere to the latency and throughput constraints of their company. As illustrated in Figure 4.1, ETUDE helps the data scientist to automate and accelerate this process. Our framework enables fully automated benchmarking, where the data scientist only needs to provide their model and declaratively specifies statistics of the underlying product catalog, hardware options for deployment (e.g., the type of GPU to use), together with latency and throughput constraints. Thereby, ETUDE reduces time-to-deployment and experimentation cost for industry practitioners. Note that ETUDE only measures the inference performance of a model, not its prediction quality; we assume that the data scientist already evaluates the prediction quality during the model design phase.

Supported models. In general, ETUDE is compatible with any SBR model implemented in PyTorch [82]. For this chapter, we discuss ten different neural SBR methods, as implemented in the open source RecBole library [128]. These include two recursive neural network-based methods: GRU4Rec [109], which utilises GRU neural networks with gating mechanisms for long-term dependencies in item interactions, and RepeatNet [93], which employs an encoder-decoder architecture with a repeat-explore mechanism.

Furthermore, we include two graph neural network-based methods: GC-SAN [123], which uses graph contextualised self-attention for session representation and SR-GNN [120], which combines graph models to predict user actions based on long-term preferences and current interests.

We use three attention-based methods: NARM [63], which employs a hybrid encoder with attention to model sequential behavior, SINE [108], which introduces sparse-interest embeddings for session recommendations, and STAMP [67], which captures short-term attention and memory using gated self-attention.

Finally, we also use three transformer-based methods: LightSANs [25], which uses transformers on session item embeddings for sequential recommendations, CORE [42], which ensures consistent session representations via weighted sum item embeddings, and SASRec [49], a self-attention-based recommendation model assigning weights to previous items in sessions.

Time complexities for inference. We derive the asymptotic time complexities for inference with the models. These complexities depend on common hyperparameters, such as the catalog size C and the number of items to recommend k. Additionally, they are influenced by model-specific hyperparameters, such as the hidden size for recursive neural networks or the embedding dimension of transformers and graph neural networks, to which we collectively refer as d. In a typical SBR scenario with a relatively short session length, the asymptotic complexity for inference with all models

Algorithm 4 Synthetic workload generation from the marginal statistics of a real clicklog.

```
1:  function GENERATE_SYNTHETIC_SESSIONS(C, N, α_l, α_c)
2:      Input: catalog size C, number of clicks N, exponents α_l and α_c for the
3:          distribution of session lengths and click counts.
4:      Output: Synthetic sessions Q.
5:      Q ← ∅
6:      n, s, t ← 0
7:      I ← sample C click counts from power law dist. with exponent α_c
8:      while n < N do
9:          s ← s + 1                                   ▷ increment session identifier
10:         l ← sample session length from power law dist. with exponent α_l
11:         n ← n + l                                   ▷ increment number of clicks generated
12:         for 0 to l do                               ▷ Generate l clicks
13:             t ← t + 1
14:             i ← sample item id from the empirical CDF of I      ▷ Choose item
15:             Q ← Q ∪ (s, i, t)                       ▷ Add synthetic click on item i in session s
        return Q
```

is $O\left(C(d + \log k)\right)$, despite the different neural architectures. This is because all models conduct a maximum inner product search for the top-k similar items in the d-dimensional learned vector representations of all C items within the catalog. The embedding dimension d is typically chosen heuristically based on the value of C and k is set to a small value, which means that the inference time is dominated by the catalog size C across all models.

Synthetic session generation. A design goal of ETUDE is to enable load testing and benchmarking without having to replay sensitive real-world click data. Therefore, we run experiments with synthetic sessions, which preserve key statistical properties of the underlying workload. Users only have to provide two statistics: the exponent α_l of a power law distribution fitted to the distribution of session lengths in the click log, and the exponent α_c of a powerlaw distribution fitted to the distribution of click counts. These statistics can be estimated once from a real click log and reused for experiments later.

We detail how to generate N synthetic clicks for a catalog with C items based on these exponents in Algorithm 4. For each synthetic session, we first sample a length l from a power law distribution with exponent α_l (Line 10), and subsequently select l items via inverse transform sampling (Line 14) from the empirical cumulative distribution (CDF) of C click counts generated upfront by sampling from a power law distribution with exponent α_c (Line 7). This algorithm is fast enough for online generation (our implementation is able to generate over one million clicks per second on a single core for a catalog size C of ten million items).

Load generator. Our goal is to measure the latency of a deployed model for a given target throughput (in terms of requests per second). However, ETUDE should still provide insightful results in situations where the model cannot handle the desired throughput (and for example times out the majority of requests). Therefore, we design a custom load generator for ETUDE, which slowly ramps up the load to a specified target throughput while keeping track of back-pressure. It will refrain from sending more

Algorithm 5 Backpressure-aware load generator, which replays the synthetic sessions Q for a target throughput r with a ramp-up over the duration d.

```
1:  function GENERATE_LOAD(r, d, Q)
2:      t, p ← 0                          ▷ Tick counter and atomic counter for pending requests
3:      for r_c ← TIMEPROP_RAMPUP(r, d) do          ▷ Main tick loop
4:          terminate in case deadline d reached
5:          t ← t + 1
6:          for i ← 0 ... r_c do                    ▷ Request generation loop
7:              terminate in case deadline d reached
8:              while p ≥ r_c do                    ▷ Backpressure handling
9:                  if no time left for current tick t
10:                     go to next tick t + 1
11:                 wait 1 millisecond
12:             if no time left for current tick t
13:                 go to next tick t + 1
14:             SCHEDULE_REQUEST_ASYNC(p, Q)
15:             d_t ← milliseconds till next tick
16:             wait for d_t/(r_c − i) milliseconds  ▷ Evenly spread out requests
17:         wait until next tick t + 1
```

work once too much back-pressure is built up, which allows us to gracefully shutdown experiments in such cases and determine the throughput threshold where a model fails to handle the load.

As detailed in Algorithm 5, the load generator ramps up the load to a target throughput r over the timespan d, while replaying the synthetic sessions Q. The load generator operates in "ticks" of one second and keeps track of the current number of pending requests. The main loop to handle a single tick starts in Line 3. In each tick, the current number r_c of requests to send per second is ramped up by the TIMEPROP_RAMPUP function, proportionally to the time spent with respect to the desired benchmark duration d, so that we reach the target throughput r eventually. Requests replay clicks from the synthetic session log Q, and are sent asynchronously (Line 14) and dynamically spread out over the duration of a tick (Line 16). During the request generation loop, the count of currently pending requests p is used to handle back-pressure: if this count reaches the current throughput target ($p \geq r_c$), the generator pauses for a millisecond to wait for the load to be handled by the server (Lines 9 and 12). Note that p is decreased when responses are received asynchronously (not shown in the pseudo code).

We implement Algorithm 5 in Java, using the asynchronous HTTP client from Apache HttpComponents 5.2.1 and integrate our synthetic clicklog generation. Our implementation additionally ensures that the load generator respects the order of the sessions, e.g., it will only send the next interaction for a session if a response for the previous interaction was received.

Inference server. We focus on efficiently serving PyTorch [82] models in ETUDE, which is the implementation framework of choice for the vast majority of state-of-the-art SBR models [128]. We spent several weeks evaluating the open source inference server TorchServe [112] for PyTorch models, which unfortunately fails to satisfy our latency and performance requirements. We experienced severe performance issues with TorchServe, which we attribute to the overhead of using several Python processes,

orchestrated by a Java frontend. We experimentally validate this finding in Section 4.3.1, where we showcase that TorchServe fails to handle 1,000 requests/second efficiently even if no model inference is performed.

As a consequence, we implement our own light-weight inference server for ETUDE in Rust, based on Actix [2], a high-performance web server leveraging non-blocking IO, the Rust bindings [111] for the C++ API of PyTorch, and a plugin enabling request batching for GPU inference [117]. Our inference server can deploy serialised PyTorch models from Google storage buckets and serve them with CPU or GPU inference. Furthermore, it allows users to configure the number of worker threads and details of the request batching. As validated in Section 4.3.1, the latency overhead of this inference server is extremely low.

Benchmark execution. We detail how to concretely execute benchmark experiments with ETUDE. We automate the cloud infrastructure management via a `make infra` command, which provisions and configures essential components such as a Kubernetes cluster, Google Storage and the addition of service accounts required for deployments. Importantly, this setup is a one-time operation, which can be reused for multiple experiments.

Experiment deployment and execution. The definition and execution of a single experiment proceeds as follows. ETUDE users declaratively specify the model(s) to deploy and the type of hardware to use. Furthermore, they specify the catalog size C, the statistics for click generation and the target throughput to which the load generator should ramp up. Subsequently, the execution is triggered via the command `make run_deployed_benchmark`. ETUDE will then deploy the model onto a dedicated machine in Kubernetes. Once the model deployment is finished (determined via Kubernetes's readiness probes), a ClusterIP service interface is deployed for allowing access to the serving machine. Next, the load generator is deployed on another machine, from which it sends the corresponding recommendation requests to the model inference server via the service interface. The load generator measures the end-to-end response latencies for its recommendation requests and the inference server additionally communicates metrics like the inference duration via HTTP response headers. The observed measurements are written to a Google storage bucket upon termination of the experiment.

4.3 Experimental Study

We validate our design decisions and showcase how ETUDE can be used to determine performant and cost-efficient deployment options for a variety of e-commerce scenarios.

If not declared otherwise, we use the following settings. We run our experiments in the Google Cloud Platform (GCP) via the Google Kubernetes Engine v1.27.3-gke.100, operating in Autopilot mode with Google Cloud SDK 442.0.0. We leverage general purpose `e2` instances [33] with 5.5 vCPUs from an Intel Xeon CPU@2.20GHz and 32 GB RAM. For the GPU experiments, we either use an NVidia-Tesla-T4 with 16GB RAM attached to an `e2` instance or a preconfigured instance with an NVidia-Tesla-A100 with 40GB GPU memory, 12 vCPUs and 85GB of RAM.

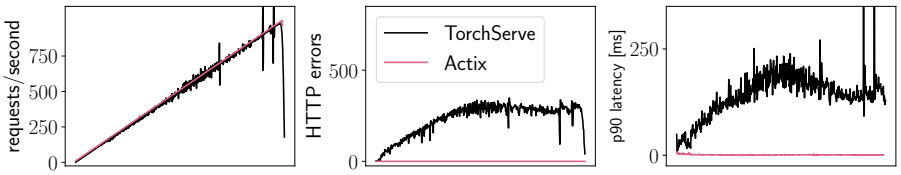


Figure 4.2: Infrastructure test for answering 1,000 requests/s without model inference. TorchServe already fails at handling "empty" requests efficiently.

We choose the embedding dimensions of the models via the common heuristic of rounding up the fourth root of the catalog size C [35] to the nearest power of two (which is in line with the original embedding sizes used in the corresponding research papers), and randomly initialise the weights of the SBR models (which need not be trained in order to measure inference performance). For synthetic session generation, we leverage marginal statistics from a real Bol click log. On our inference server, we apply request batching for GPUs for up to 1,024 requests, and empty the underlying buffer every two milliseconds. We make our code and experimental results available.[3]

4.3.1 Validation of Design Choices

First, we validate our design choice of leveraging an Actix-based Rust runtime instead of the open source TorchServe project as serving engine.

Experimental setup. In order to measure the serving performance of TorchServe independent of the model inference overhead, we deploy TorchServe on a 2 vCPU `e2` machine with 2GB of memory, and implement a Python model that returns a empty response and does not conduct any computation. Next, we configure our load generator to ramp up to 1,000 requests per second over the duration of ten minutes, and measure the response latencies. We deploy our Actix-based inference server analogously and also make it return a static answer.

Results and discussion. We plot the results of our experiment in Figure 4.2. The load ramps up to 1,000 requests per second over 10 minutes, and we observe early on that TorchServe cannot keep up with the load and starts to return a large number of HTTP errors (due to reaching the internal timeout of 100ms). It handles the remaining requests with a p90 latency between 100 and 200ms. Our Actix-based inference server easily handles the load with a p90 latency of around one millisecond for serving the static content and does not throw any HTTP errors. These results are a strong indication that TorchServe's design causes severe latency overheads and that it is not suitable for low-latency, high throughput use cases like session-based recommendation. This insight is further supported by the documentation on benchmarking and tuning TorchServe, which only uses workloads with a small number of requests (1,000 in total) and low concurrency (10 requests at the same time) [86, 105].

We also run a validation experiment for the synthetic click generation, where we compare the latency measurements achieved by replaying a real click log from Bol

[3] https://github.com/bkersbergen/etudelib/blob/main/experiments.md

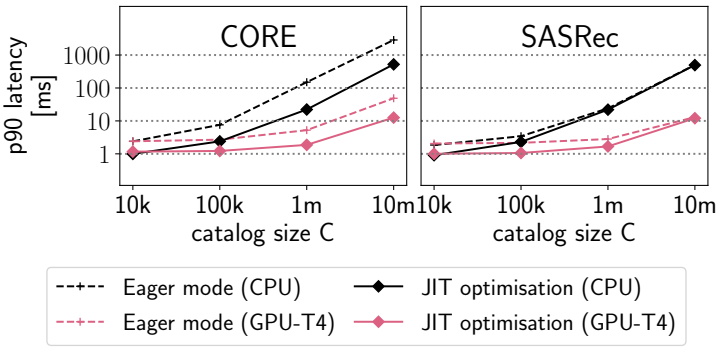


Figure 4.3: Microbenchmark results confirming the dependency of the inference latency on the catalog size and the benefits of applying PyTorch's JIT optimisations.

to the measurements achieved when using a synthetic workload generated based on statistics from the real click log. We find that the achieved latencies resemble each other closely.

4.3.2 Micro-Benchmark

We run a micro-benchmark to confirm our theoretical insight about the impact of the catalog size on prediction latency, the potential of accelerator hardware and the benefits of PyTorch's just-in-time (JIT) optimisation [87].

Experimental setup. We conduct our micro-benchmark on a single machine with synthetic click data generated according to the marginal statistics of session lengths from Bol and varying catalog sizes. We send recommendation requests in a serial manner (one request after another, waiting for model responses), measure the prediction time and report the p90 latency. We repeat the experiment with two different instances (CPU and GPU-T4), different execution types (eager execution without optimisation and JIT-optimised versions of the models) and catalog sizes of 10,000, 100,000, 1,000,000, and 10,000,000 distinct items.

Results and discussion. We observed similar results for all models and plot a selection of the resulting prediction latencies in relation to the catalog sizes on a logarithmic scale in Figure 4.3. The results confirm our theoretical analysis from Section 4.2 about the strong dependence of the runtime on the catalog size, as we observe a linear scalability of the prediction latency with the catalog size.

We see clear benefits of using GPUs for medium to large catalog sizes: starting from catalogs with one million items, the prediction latency of the GPU is more than an order of magnitude lower than the latencies achieved with CPUs only (and the CPU already requires more than 50ms per prediction for catalogs with one million items). Interestingly, this relation does not hold for small catalogs with 10,000 items, in six out of ten cases, the CPU latency is on par with or lower than the GPU latency here. Furthermore, we find that JIT-optimisation is always beneficial and never hurts performance. We identify an issue with the LightSANs model implementation though,

Table 4.1: Cost-efficient deployment options for SBR models in varying e-commerce scenarios with costs per month, derived from ETUDE measurements. Boldface indicates the most cost-efficient deployment option for a scenario. Empty cells for models indicate that they are not able to handle the target throughput with the given deployment option.

Scenarios			Deployment Options			SBR Models					
Use case	Catalog size	Throughput	Instance type	Number	Cost/ month	CORE	GRU4Rec	NARM	SASRec	SINE	STAMP
Groceries (small)	10,000	100 req/s	**CPU**	**1**	**$108**	✓	✓	✓	✓	✓	✓
			GPU-T4	1	$268	✓	✓	✓	✓	✓	✓
Groceries (large)	100,000	250 req/s	**CPU**	**1**	**$108**	✓	✓	✓	✓	✓	✓
			GPU-T4	1	$268	✓	✓	✓	✓	✓	✓
Fashion	1,000,000	500 req/s	CPU	3	$324				✓		✓
			GPU-T4	**1**	**$268**	✓	✓	✓	✓	✓	✓
			GPU-A100	1	$2,008	✓	✓	✓	✓	✓	✓
e-commerce	10,000,000	1,000 req/s	**GPU-T4**	**5**	**$1,343**		✓	✓	✓	✓	✓
			GPU-A100	2	$4,017	✓	✓	✓	✓	✓	✓
Platform	20,000,000	1,000 req/s	**GPU-A100**	**3**	**$6,026**		✓	✓		✓	✓

which cannot be JIT-optimised by PyTorch due to dynamic code paths.

4.3.3 End-to-End Benchmark

Our goal for the final experiment is to showcase how ETUDE can identify well performing models and cost-efficient deployment setups for a variety of e-commerce use cases.

Experimental setup. We define five end-to-end use case scenarios, detailed in the first three columns of Table 4.1: *Grocery shopping (small)*, *Grocery shopping (large)*, *Fashion*, *e-commerce*, and *Platform* with catalog sizes from 10,000 up to 20,000,000 items and a target throughput ranging from 100 to 1,000 requests per second. These scenarios are inspired by experiences from our various brands and use cases at Ahold Delhaize, and are in line with publicly reported catalog sizes [3, 11, 127].

We conduct an end-to-end benchmark for the JIT-optimised variants of all ten models in all scenarios with three different instance types (CPU, GPU-T4, and GPU-A100). We ramp up the load to 1,000 requests per second over a period of ten minutes and measure the response latency. We execute each configuration three times and ignore the runs with the lowest and highest latencies, amounting to around four hundred runs.

Results and discussion. We plot detailed results for a selection of scenarios in Section 4.4 and discuss various aspects of our findings.

Issues with selected SBR models. We encounter serious issues with three additional SBR model implementations from the RecBole library [128]: SR-GNN, GC-SAN and RepeatNet are not able to handle most of our use cases (or only handle them with unacceptably low performance). We inspect their implementations to determine

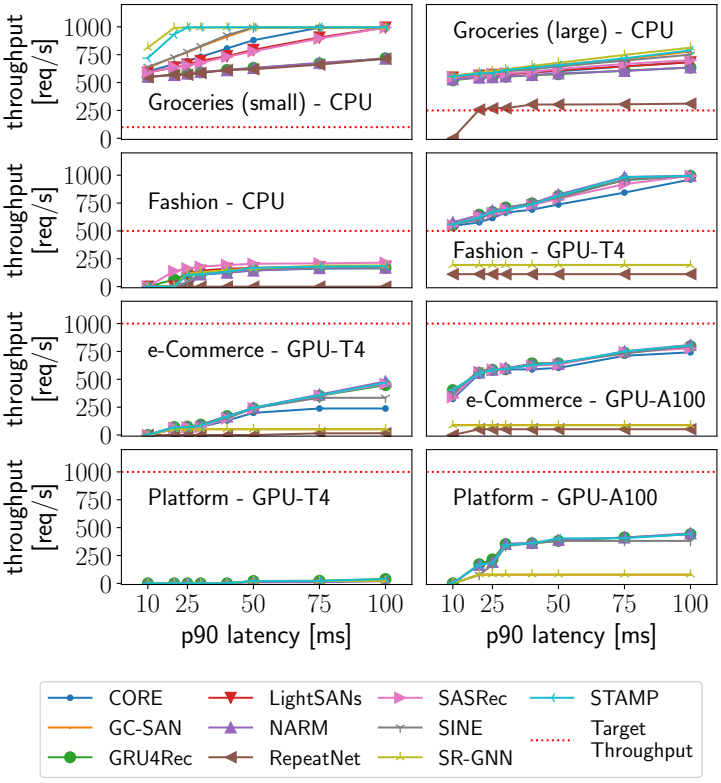


Figure 4.4: Observed latency and throughput of different SBR models in deployment scenarios with varying instances types.

the root causes for this finding. The RepeatNet model contains expensive tensor multiplications of very sparse matrices which are implemented with dense operations and representations (and therefore incur high overheads), and the SR-GNN and GC-SAN models contain NumPy operations in their inference functions which require repeated data transfers between CPU and GPU at inference time. We filed bug reports [97] for these issues with the RecBole project.

Impact of catalog size and accelerator hardware. We observe that catalog sizes of 10,000 and 100,000 can be handled well with CPU instances only, where most models achieve more than 500 requests per second at a 50ms p90 latency. The situation changes for catalogs with one million items, where the performance of CPU instances drops to around 200 milliseconds. At the same time, we see that this setup is easily handled by instances with GPUs, where the T4 card already handles more than 700 requests per second at a 50ms p90 latency. Only GPU instances are able to handle the load for catalogs with 10 million items, and for the platform setting with 20 million items, the high-end A100 cards are required.

Cost-efficient deployment options. The most performant setup may not necessarily be

the most cost-efficient one. The monthly costs (given a one year commitment) for different machines vary [34]; a CPU instance, for example, costs $108.09 in GCP, an instance with an additional T4 GPU costs $268.09 per month, and the instance with the A100 GPU has a hefty price tag of $2,008.80. There may be cases where it is more beneficial to linearly scale out the recommendation system with cheaper hardware than to use a high-end device.

Such cost decisions can be made based on ETUDE's experimental results: Table 4.1 lists the monthly costs for the best models per scenario for different setups. Note that we applied a latency threshold of 50 milliseconds in the 90th quantile and ignored the four models for which we found implementation errors. We find that (i) both grocery shopping scenarios can be handled very cost-efficiently with a single CPU machine for $108 per month (for all models); (ii) the specialised e-commerce scenario can be handled with a single GPU-T4 instance (for all models) for $268 per month, and two models (SASRec and STAMP) are also comparatively cheap to deploy with only CPUs (at a cost of $324 per month); and (iii) the general e-commerce and platform scenario require GPUs: the platform scenario with a large catalog of 20 million items can only be efficiently handled with three high-end GPU-A100 instances at the high cost of $6,026 per month. Interestingly, for the general e-commerce scenario, it is significantly cheaper to deploy five GPU-T4 instances ($1,343) than to leverage two more powerful GPU-A100 instances (for $4,017).

4.4 Conclusion

We presented ETUDE, a framework to automatically evaluate the inference performance of SBR models under declaratively specified deployment options in terms of hardware, workload statistics and latency and throughput constraints.

In the past, we have seen recommendation teams refrain from building online Session-Based recommendation systems due to the outlined serving challenges. As a consequence, they designed static recommendation systems with precomputed recommendations for the last item of a session only, which often exhibit low prediction quality due to the missing session context. ETUDE is currently helping such teams to reduce risk, as they can test newly designed models early on challenging workloads, and to improve their model implementations by identifying bugs which impact performance.

In the future, we plan to extend ETUDE with more inference runtimes such as ONNX [10] or TensorRT [79] and to support additional cloud environments such as Microsoft Azure or Amazon Web Services. Furthermore, we will explore the incorporation of techniques to trade-off prediction quality with inference latency, such as model quantisation [31] or approximate nearest neighbor search [48], as well as the automatic choice of appropriate instance types for declaratively specified workloads. Our findings also indicate that there is a need to design custom neural models for high cardinality catalogs. This is evidenced by the enormous costs for deploying models on catalogs with twenty million items, which can be handled much cheaper with non-neural approaches [52].

In the next chapter, we shift our focus to the data aspect of recommendations, exploring how Data Shapley values can be used to debug real-world interaction datasets

for sequential kNN-based recommendation systems in order to improve recommendation quality.

5

Illoominate – Scalable Debugging of Recommendation Data in e-Commerce

Modern e-commerce platforms offer millions of items to their customers. The challenge of navigating such vast collections has led to the development of machine learning-powered recommendation systems that help users find items that they might like. These recommendation systems predict future interactions between users and products by being trained on data about such interactions that were recorded in the past. Failures of such systems can severely impact the user experience, such as recommending dangerous items, and can often be directly attributed to issues present in the training data, also called *data errors*. Unfortunately, given the massive scale of these datasets, the process of identifying the data errors that have the most significant negative impact, also called *data debugging*, can be prohibitively expensive.

In this chapter, we present ILLOOMINATE, a library for the scalable debugging of interaction data for our recommendation systems at Bol, one of the largest European e-commerce platforms. Our library assists data scientists in uncovering various data errors via recent data importance algorithms and scales to datasets with millions of interactions.

We therefore formulate our next research question:

RQ4 How can we efficiently compute Data Shapley values for sequential kNN-based recommendation systems on real-world datasets with millions of datapoints?

We answer this question by designing the KMC-Shapley algorithm, a custom-tailored, scalable variant of a recent Monte Carlo-based algorithm for estimating so-called Data Shapley values. This is achieved by leveraging the special characteristics of our KNN-based recommendation models in production.

Our next research question is:

RQ5 Are Data Shapley values helpful for debugging real-world interaction data in sequential kNN-based recommendation systems?

We conduct an experimental evaluation of the efficiency and scalability of ILLOOMINATE on both public and proprietary datasets for session-based and next-basket rec-

This chapter is under submission as B. Kersbergen, O. Sprangers, B. Karlaš, M. de Rijke, and S. Schelter. Illoominate – Scalable debugging of recommendation data in e-commerce. Under submission, 2025.

ommendation, and showcase its ability to reliably identify the most severe data errors for each of these tasks. Moreover, we discuss various applications of our library on click and purchase data from Bol. These applications include discovering dangerous products and low-quality products, as well as increasing the ecological sustainability of recommendations via data pruning.

5.1 Introduction

In recent years, online shopping has moved to e-commerce platforms, where millions of items are offered by thousands of third-party sellers on centralised marketplaces. Bol, one of Europe's largest e-commerce platforms,[1] for example, features 42 million products offered by more than 47,000 external partners to 25 million active users. The challenge of navigating such vast collections has led to the development of recommendation systems powered by machine learning (ML) [66], which help users find items that they might like. Common tasks for recommendations on e-commerce platforms are session-based recommendation [52, 70], where the goal is to predict the next item that a user will click on, or next-basket recommendation [6, 43, 64], where the goal is to predict the items in the next shopping basket purchased by a user.

Data errors impeding e-commerce recommendation systems. Real-world recommendations are complex systems, operating on data that captures interactions between users and products. The unpredictable nature of this data frequently produces *data errors* that are the root cause of many system failures, e.g., the accidental recommendation of dangerous items [14, 78], externally manipulated rankings [12], or third-party products with low-quality metadata [7, 115]. Furthermore, the data collection process is exposed to various sources of noise, e.g., "multi-journeys," where a single user shops for multiple things simultaneously (such as presents for family members of different age groups), or "unnatural interaction patterns" caused by bots crawling the website to record the product assortment and prices.

To make matters worse, as these issues are hard to anticipate upfront, they are typically detected post-deployment after they have already damaged the user experience. For example, we recently faced a situation where sensitive adult items were included in the recommendations on product pages of children's toys, due to the unexpected co-occurrence of these types of products in some historical browsing sessions. This incident prompted an urgent live patch of the system with custom filtering rules, followed by the manual identification and removal of the session data in which these unwanted co-occurrences appeared.

Data debugging via data importance. Preventing such predicaments requires a principled *data debugging* process during which human experts work to identify data errors and preemptively mitigate them. However, the scale of our training datasets renders this process prohibitively expensive. One way forward would be to offer systematic guidance highlighting the data points that are more likely to be causing issues. As part of the recent data-centric initiative of the ML community [80], various notions of *data importance* [32, 37, 50, 61, 118] have been proposed to quantify the

[1] https://over.bol.com/en/about-bol/#facts-and-figures, accessed in November 2024.

impact of individual training data points on an ML model (see Section 5.3.1 for details). The most prominent such notion is the *Data Shapley value (DSV)* [32], which has already attracted considerable interest in the data management community [72] and has been shown to be effective at certain data debugging tasks [47, 50].

However, prior work on evaluating the DSV for importance-driven data debugging mainly focuses on classification tasks. To the best of our knowledge, no study has evaluated its effectiveness when applied to recommendation tasks. Furthermore, current approaches for overcoming the exponential complexity of computing the Shapley value are not directly applicable to recommendation models. Some methods treat the entire model as a black box [32, 61] which could only be feasible for datasets that are orders of magnitude smaller than ours. Other methods make assumptions about the additivity of model quality metrics [46, 50] that are not directly applicable to recommendation systems. In this work, we take a step towards closing this gap and expanding the range of possible scenarios that could benefit from principled data debugging approaches. Specifically, we explore the following questions:

1. *Can we efficiently compute Data Shapley values for recommendation systems on real-world datasets with millions of interactions?*

2. *Are Data Shapley values helpful for debugging real-world interaction data in recommendation systems?*

Overview and contributions. We present ILLOOMINATE, a library for the scalable computation of data importance for interaction data from our recommendation systems at Bol (Section 5.4). ILLOOMINATE is implemented in Rust with a simple Python frontend, scales to datasets with millions of interactions, and makes it easy for data scientists to run data importance algorithms and leverage the results in downstream applications.

In particular, we explore the potential of the Data Shapley value for debugging real-world interaction data of our production recommendation system *Serenade* [52] for session-based recommendation (presented at SIGMOD'22) and for next-basket recommendation models [6, 43] deployed on European online grocery platforms of our partner companies [114]. Apart from data debugging, we also show how to leverage data importance to prune the training data of one of our recommendation systems to increase the number of sustainable items in its predictions, without decreasing recommendation quality.

In summary, in this chapter we provide the following contributions.

Contribution 1: Design of ILLOOMINATE *(Section 5.4).* We detail the design of our library ILLOOMINATE for debugging large-scale recommendation data via data importance. We make our code available at `https://github.com/bkersbergen/illoominate`.

Contribution 2: KMC-Shapley algorithm (Section 5.5). We exploit the fact that our recommendation systems are built on a special class of models, namely nearest neighbor models for sequential recommendation, and that the interaction data in real-world recommendation systems is extremely sparse. Building on these characteristics, we design the KMC-Shapley algorithm, a custom-tailored, scalable variant of a recent

Monte Carlo-based algorithm for estimating the Data Shapley Value, and implement it as part of our library.

Contribution 3: Experimental evaluation (Section 5.6). We conduct an experimental evaluation of the efficiency and scalability of ILLOOMINATE on both public and proprietary datasets with millions of interactions. Additionally, we adapt experiments from academic papers to showcase that the DSV identifies impactful data points for session-based and next-basket recommendation tasks.

Contribution 4: Discussion of Applications (Section 5.7). Finally, we discuss applications of ILLOOMINATE on click and purchase data from Bol. These applications include identifying dangerous and low-quality products as well as improving of the ecological sustainability of recommendations via data pruning. Furthermore, we provide initial evidence that our computed DSVs are also meaningful for neural recommendation models.

5.2 Related Work

To the best of our knowledge, we are the first to explore data importance for recommendations in e-commerce. We list related work across the areas underlying this direction.

Recommendation systems. Recommendation systems [15, 16, 28, 36, 38, 73, 83, 94, 107, 124, 129, 131, 132] are an active area of research, with a close connection to industry use cases [113, 119, 122]. A particular challenge is to translate academic progress into deployable solutions at scale [13, 44, 51–53, 62].

Error detection for ML. Detecting errors in data is a core research direction in data management [1, 65], and several approaches tailored to ML applications have been proposed. Google TFX [84] and Deequ [100, 102] infer integrity constraints for ML data based on data profiling, and follow-up approaches learn to validate ML data from historically observed data partitions [92, 104]. HoloDetect [39] and Picket [68] detect erroneous samples in classical supervised learning scenarios via few-shot learning and self-supervised learning.

Data importance. Determining the importance of a data point for an ML model is a core question in data-centric AI [37], with many applications in data debugging and data selection. Prominent notions of data importance include the Data Shapley value [32], and generalisations such as the Beta Shapley value [61] and the Data Banzhaf value [118]. The efficient estimation of such data importance scores is an active area of research with promising results for special classes of models such as KNN classifiers [46], which can be used as a proxy for more expensive models. Datascope [50] and ArgusEyes [103] use Data Shapley values to debug the outputs of ML pipelines, while Gopher [85], Rain [27, 121] and follow-up work [22] leverage estimates of the leave-one-out error computed via influence functions [55] for data debugging.

Besides ML-specific scenarios, there is a wide variety of applications of the Shapley value in data management; see [72].

5.3 Background

We introduce data importance and sequential recommendation as foundations for the remainder of this chapter.

5.3.1 Data Importance

The goal of data importance (often also referred to as data valuation) is to quantify the impact of each individual training data point on the quality of a machine learning model [37]. Different notions of this importance [32, 61, 118] have been proposed, which all rely on measuring the impact of excluding a given data point from the model's training data (or certain subsets of it). More formally, let $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ denote the set of n training data points whose importance we want to compute. Furthermore, let the *utility function* $V(S)$ measure the performance of a model trained on a subset $S \subseteq \mathcal{D}$ of the training data. For example, V might compute the recall of a recommendation model on held-out data $\mathcal{D}_{val}$. We discuss the two most prominent notions of data importance here:

Leave-One-Out error (LOO). The simplest way to measure the importance of a data point $\mathbf{x}_i \in \mathcal{D}$ is its leave-one-out error, the change in utility V when the data point $\mathbf{x}_i$ is excluded from the training data $\mathcal{D}$:

$$V(\mathcal{D}) - V(\mathcal{D} \setminus \{\mathbf{x}_i\}). \tag{5.1}$$

While the LOO is easy to compute, it is empirically found to be highly noisy [61] and typically outperformed by more complex notions of importance in its helpfulness for downstream tasks [37].

Data Shapley value (DSV). Recently, the Data Shapley value (DSV) has been proposed as an equitable way to measure the importance of training data points for a machine learning model [32]. The DSV determines the "value" ϕ_i of each data point $\mathbf{x}_i$ from the data $\mathcal{D}$ based on the well-known Shapley value from game theory [60]:

$$\phi_i = \frac{1}{n} \sum_{S \subseteq \mathcal{D} \setminus \{\mathbf{x}_i\}} \binom{n-1}{|S|}^{-1} V(S \cup \{\mathbf{x}_i\}) - V(S). \tag{5.2}$$

The DSV measures the weighted average of the marginal contribution $V(S \cup \{\mathbf{x}_i\}) - V(S)$ of adding the data point $\mathbf{x}_i$ to all $2^{|\mathcal{D}|-1}$ subsets S of the training data $\mathcal{D}$. The DSV is challenging to compute, but has been shown to work well empirically, e.g., for tasks like detecting mislabeled data [32, 50, 61, 103].

5.3.2 Sequential Recommendation

The general goal of *sequential recommendation* (SR) is to predict the next action in a sequence of user actions; it subsumes many core recommendation tasks in e-commerce. In *session-based recommendation* (SBR), the goal is to predict the next product that a user browsing the website will visit, based on the sequence of already visited products [40, 52, 70, 123]. In *next-basket recommendation* (NBR), the task is

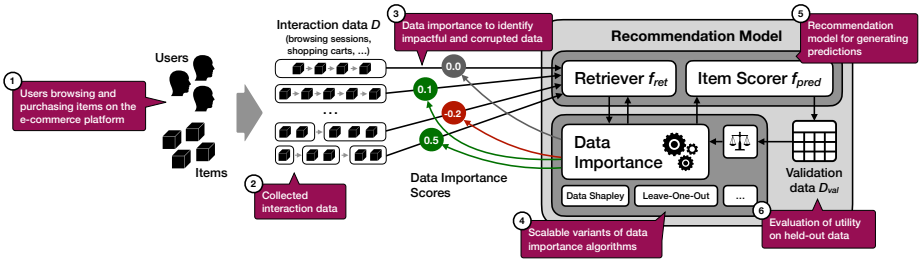


Figure 5.1: Overview of ILLOOMINATE – our library operates on data originating from interactions between customers and items on our e-commerce platform ❶, such as browsing sessions or shopping baskets, which may contain various data issues ❷. ILLOOMINATE computes data importance scores ❸ to identify impactful and potentially corrupted data points. For that, we design efficient variants of common data importance algorithms ❹, specialised to our recommendation systems, such that they scale to millions of interactions. During the execution of these algorithms, various data subsets are constructed and the utility of a KNN-based recommendation model ❺ trained on these subsets is evaluated for held-out validation data ❻.

to predict the set of products that a user will purchase in their next shopping basket, based on the sequence of already purchased shopping baskets from the past [43, 64]. A variant of next-basket recommendation is *within-basket recommendation* [6], where a given set of items is already in the current shopping basket.

Formally, we define SR as follows. Let $\mathbf{s} = [c_0, \ldots, c_t]$ denote a sequence of interactions with items $c \in \mathcal{I}$. Such an interaction sequence may, for example, represent clicks on product pages during a browsing session on a web platform or a shopping basket purchased on an e-commerce platform. Given the history of interaction sequences $\mathbf{h}_u = [\mathbf{s}_0^u, \ldots, \mathbf{s}_n^u]$ for each user $u \in \mathcal{U}$, common tasks are to predict either the whole next interaction sequence $\mathbf{s}_{n+1}^u$ (e.g., the next shopping basket that a user will purchase) or to predict the next item i_{t+1} of an evolving interaction sequence $\mathbf{s}_{n+1}^u = [i_0, \ldots, i_t]$ (e.g., the next product a user will click on during a visit on the website). As output, SR models predict a ranked list of items and the predictions are evaluated with common rank-based metrics such as Mean Reciprocal Rank (MRR), Recall or Normalised Discounted Cumulative Gain (NDCG) [8]. Note that scenarios without user information are a special case of the general task outlined here.

In recent years, a set of nearest neighbor-based approaches to SR have been proposed [6, 24, 29, 43, 52, 70]. These models, to which we refer as KNN-SR models, compute sparse representations of the user histories in the training data and use them as a retrieval corpus at inference time. The recommendations for a query user are based on the k most similar retrieved neighbors. Despite their simplicity, such KNN-based approaches to sequential recommendation provide state-of-the-art performance [6, 26, 43, 45, 64] and are deployed in large-scale production scenarios with millions of users [52, 114]. Our product recommendation system at Bol, which we presented at SIGMOD'22 [52], also leverages such a model. We refer to Section 5.9.1 for details on the KNN-SR models that we use in production.

5.4 ILLOOMINATE

We detail the design of our library ILLOOMINATE for the scalable computation of data importance for recommendation data.

Overview. Figure 5.1 details the high-level design of ILLOOMINATE – our library operates on data originating from interactions between customers and items on our e-commerce platform ❶, such as browsing sessions or shopping baskets, which may contain various data issues ❷. ILLOOMINATE computes data importance scores ❸ to identify impactful and potentially corrupted data points. For that, we design efficient variants of common data importance algorithms ❹, specialised to our recommendation systems, such that they scale to millions of interactions. During the execution of the data importance algorithms, various data subsets are constructed and the utility of a KNN-based recommendation model ❺ trained on these subsets is evaluated for held-out validation data ❻.

ILLOOMINATE is implemented in Rust, with a Python frontend to invoke it within data science code. In order to compute data importance scores, users provide their interaction data with a validation set, and specify the parameters of the recommendation scenario. For example, potentially corrupted session data (with negative DSVs) can be identified as follows:

```
import pandas as pd
import illoominate as ilm

sessions = pd.read_parquet(...)
validation = pd.read_parquet(...)

# Data Shapley values for sessions
data_shapley_values = ilm.data_shapley_values(
    train_df=sessions, validation_df=validation, model="vmis",
    metric="mrr@20", params={"m":250, "k":250, "seed":42})

# Problematic sessions with negative data shapley values
negative = data_shapley_values[data_shapley_values.score < 0.0]
corrupted_sessions = sessions.merge(negative, on="session_id")
...
```

Interaction data. Our library currently supports two types of interaction data: (i) *browsing sessions*, consisting of a session identifier and a list of time-stamped clicks on items; and (ii) *purchase histories*, consisting of a user identifier and a list of purchased shopping baskets, each represented by a set of item identifiers. This data is provided in the form of dataframes or CSV files, which follow a simple schema: for sessions, each line represents a click with a `session_id`, `item_id` and `timestamp`; for purchase histories, each line represents a purchase event with a `user_id`, `basket_id` and `item_id`.

Recommendation models. We specialise our library to KNN-SR models (Section 5.3), as we run several of these models in production for session-based recommendation and next-basket recommendation [52, 114]. An important design goal of our library is to abstract from the concrete instantiation of a particular KNN-SR algorithm so that we can implement data importance algorithms once and apply them to all models from the KNN-SR family. In order to achieve this, we define an abstract computational model capturing the general "mechanics" of KNN-SR algorithms. In the following, we discuss

the components of this computational model (which are instantiated differently for different models).

Sparse representation of interaction histories. Each user's interaction history $\mathbf{h}_u$ is encoded as a sparse representation $\mathbf{x}_u$. Thereby, the training data of observed user interactions is turned into a retrieval corpus $\mathcal{D} = \{\mathbf{x}_u \mid u \in \mathcal{U}\}$.

Retrieval-based inference. KNN-SR models consist of a retrieval function f_{ret} to query the retrieval corpus $\mathcal{D}$ and a prediction function f_{pred} to generate recommendations based on the retrieved representations. At inference time, KNN-SR models compute recommendations for a query user q in two stages:

1. *Retrieval* – The k nearest neighbors $\mathbf{x}_{\alpha_1}, \ldots, \mathbf{x}_{\alpha_k}$ for the query user representation $\mathbf{x}_q$ are retrieved via $f_{ret}(\mathbf{x}_q, \mathcal{D}, k)$, according to a model-specific ranking function.

2. *Item scoring* – The top items to recommend to user q are computed via the prediction function $f_{pred}(\mathbf{x}_q, \{\mathbf{x}_{\alpha_1}, \ldots, \mathbf{x}_{\alpha_K}\})$, based on the query user representation $\mathbf{x}_q$ and the retrieved representations $\mathbf{x}_{\alpha_1}, \ldots, \mathbf{x}_{\alpha_k}$.

We refer to Section 5.9.1 for details on the concrete mapping of our deployed KNN-SR algorithms to this model. In our Rust code, we define the trait `RetrievalBasedModel` for all recommendation algorithms in ILLOOMINATE based on the outlined computational model. We implement the VMIS-kNN [52] and TIFU-kNN [43] algorithms accordingly.

Data importance. ILLOOMINATE currently supports computing data importance for interaction data via Data Shapley values and leave-one-out errors (as detailed in Section 5.3.1). All data importance algorithms are implemented in a general way via a Rust trait called `Importance` which is compatible with any recommendation model supporting the previously defined `RetrievalBasedModel` trait. We also design a general trait for utility functions and implement common ranking-based metrics in recommendation systems such as Mean Reciprocal Rank (MRR), Normalised Discounted Cumulative Gain (NDCG), Recall, Precision or HitRate [8]. Our design also allows users to implement custom utility functions for their use cases.

5.5 KMC-Shapley

The main challenge for ILLOOMINATE is the scalable computation of Data Shapley values for our recommendation systems. In the following, we discuss the underlying scalability issues and detail the KMC-Shapley algorithm, which we design to address them.

5.5.1 Scalability Issues

Unfortunately, computing exact Data Shapley values, as defined in Section 5.2, is intractable in practice because the number of subsets to process is exponential in $|\mathcal{D}|$.

Algorithm 6 TMC-Shapley algorithm as proposed in [32].

1: **function** TMC-SHAPLEY($\mathcal{D}, V$)
2: $\phi = \{\phi_1, ..., \phi_n\} \leftarrow 0$
3: **while** not converged **do**
4: $\pi \leftarrow$ random permutation of data points in $\mathcal{D}$
5: $v_{\text{prev}} \leftarrow V(\emptyset)$
6: **for** $j \in 1 \dots n$ **do** ▷ Iterate through permutation
7: **if** $|V(D) - v_{\textbf{prev}}| >$ **performance tolerance** ▷ Truncation
8: $S_\pi^i \leftarrow$ set of data points before position j in π
9: $\mathbf{x}_i \leftarrow$ data point at position j in π
10: $v \leftarrow V(S_\pi^i \cup \{\mathbf{x}_i\})$ ▷ Retrain and evaluate model
11: $\phi_i \leftarrow \phi_i + (v - v_{\text{prev}})$ ▷ Update DSV with marginal contribution
12: $v_{\text{prev}} \leftarrow v$
13: $t \leftarrow$ number of permutations evaluated
14: $\phi \leftarrow \frac{1}{t}\phi$ ▷ Normalise by number of iterations
15: **return** Shapley values $\phi = \{\phi_1, \dots, \phi_n\}$

Even worse, each subset S to process requires the re-training and evaluation of the underlying recommendation model.

TMC-Shapley. Computing the Data Shapley value ϕ_i for a data point $x_i \in \mathcal{D}$ can be formulated as an expectation calculation problem: $\phi_i = \mathbb{E}_{\pi \sim \Pi}\left[V(S_\pi^i \cup \{\mathbf{x}_i\}) - V(S_\pi^i)\right]$, where Π denotes the uniform distribution over all $n!$ permutations of the data points in $\mathcal{D}$ and S_π^i is the set of all data points that come before $\mathbf{x}_i$ in the permutation π [32]. For this formulation, Ghorbani et al. [32] present the TMC-Shapley algorithm (Algorithm 6), which conducts a Monte Carlo (MC) estimation of the Data Shapley values.[2] TMC-Shapley repeatedly samples a permutation π from Π, computes the marginal contribution $V(S_\pi^i \cup \{\mathbf{x}_i\}) - V(S_\pi^i)$ of a data point $\mathbf{x}_i$ over S_π^i in Lines 10 and 11, and iterates through all n data points in the permutation (Line 6). Furthermore, it applies a truncation technique, by stopping to compute marginal contributions once the obtained utility is within a threshold of the utility value $V(D)$ of the whole dataset (Line 7). A convergence test is performed every 100 iterations by checking if the mean absolute percentage deviation of the Shapley value for all data points is below a given threshold.

Scalability issues in TMC-Shapley. Even Monte Carlo-based algorithms such as TMC-Shapley are difficult to scale to larger datasets [61], as the number of models to retrain and evaluate is still linear in the number of data points in $\mathcal{D}$. This effort is not feasible for large datasets and many modern models, which is exemplified by the fact that the experiments in [32, 61] are conducted on hundreds of data points only and do not even fully retrain models such as neural networks (where only the last layer is retrained).

Another line of research details how to efficiently compute DSVs for KNN classifiers [46] (which can act as proxies for other models). This approach makes several assumptions about the utility function that prevent it from being directly applicable to KNN-based recommendation systems. Firstly, it assumes that the model quality metric

[2] Note that we base Algorithm 6 on the author's actual code from https://github.com/amiratag/DataShapley, which slightly differs from the formulation in their paper.

treats the contributions of the nearest neighbors independently, which is not the case for ranking-based metrics used in recommendation systems. Secondly, its computational efficiency bounds are derived for classification tasks with a limited number of class labels, which is not the case in recommendation systems, which have to rank millions of items.

5.5.2 Data and Model Characteristics in KNN-SR

Our KMC-Shapley algorithm is a specialised variant of TMC-Shapley, which skips the expensive utility computation in cases where we already know that the marginal contribution is zero. Before presenting the actual algorithm, we first discuss the data and model characteristics in KNN-SR which enable us to do this.

Sparse retrieval. SR algorithms operate on interaction data from online platforms. A common characteristic of these platforms is that they offer a wide selection of items, while each user interacts only with a tiny fraction of the available items. For example, the median session length in SBR datasets ranges from two to four clicks, even in scenarios with more than a million distinct items [52], and shopping baskets in NBR datasets contain less than ten items on average, even though there are tens of thousands of distinct items available [43]. As a consequence, the resulting interaction data is extremely sparse. To account for this, KNN-SR algorithms leverage sparse representations for the interaction histories and apply a sparse retriever f_{ret} with a similarity function $d(\mathbf{x}_q, \mathbf{x}_i)$ to rank a representation $\mathbf{x}_i$ in the retrieval corpus with respect to a query $\mathbf{x}_q$. These retrievers typically ignore pairs $(\mathbf{x}_q, \mathbf{x}_i)$ of representations with no overlap in item interactions, meaning that $\mathbf{x}_i$ will never be in the neighbor set of $\mathbf{x}_q$ in that case.

Locality. The prediction function f_{pred} only uses the k representations with the largest similarities returned by f_{ret}. Let $\Gamma_q(S)$ denote the k-th largest similarity score between the validation sample $\mathbf{x}_q$ and the representations from a set S. If $d(\mathbf{x}_q, \mathbf{x}_i) < \Gamma_q(S)$, then adding $\mathbf{x}_i$ to S has no impact on the utility for $\mathbf{x}_q$.

The sparsity of the interaction data and the locality property of KNN models allow us to work with the much smaller set of neighbors of each validation sample instead of having to process all data points from the training data in each iteration. Apart from being able to skip certain utility computations, we can further accelerate the algorithm based on the following characteristics.

Additivity of utilities. Analogous to classification and regression scenarios, the utility V of a model with respect to a validation dataset $\mathcal{D}_{val}$ in SR is computed as the sum of the utilities for the individual validation samples $\mathbf{x}_q \in \mathcal{D}_{val}$, leading to $V(S_\pi^i \cup \{\mathbf{x}_i\}) - V(S_\pi^i) = \sum_{\mathbf{x}_q \in \mathcal{D}_{val}} V_q(S_\pi^i \cup \{\mathbf{x}_i\}) - V_q(S_\pi^i)$, where V_q computes the utility with respect to the validation sample $\mathbf{x}_q$. This additivity of utilities enables a mapreduce-like parallelisation pattern, where we process each individual validation sample $\mathbf{x}_q \in \mathcal{D}_{\text{val}}$ in parallel and aggregate the resulting marginal contributions per training data point subsequently.

Linear aggregations at prediction time. The prediction function f_{pred} in KNN-SR models typically first conducts a linear aggregation of the retrieved representations

Algorithm 7 KMC-Shapley – custom-tailored, scalable variant of TMC-Shapley specialised to KNN-SR models.

function KMC-SHAPLEY($\mathcal{D}, V, k, \mathcal{D}_{\text{val}}$)
  $\phi = \{\phi_1, \ldots, \phi_n\} \leftarrow 0$ ▷ Initialize Data Shapley values
  $\mathbf{N} \leftarrow \emptyset$ ▷ Initialize neighbor index
  for $\mathbf{x}_q \in \mathcal{D}_{\text{val}}$ **do** ▷ Populate index with neighbors of validation samples
    $\mathbf{N}_q = \{(\mathbf{x}_{\alpha_1}, d_{\alpha_1}), \ldots, (\mathbf{x}_{\alpha_M}, d_{\alpha_M})\} \leftarrow f_{\text{ret}}(\mathbf{x}_q, \mathcal{D})$
  while not converged **do**
    $\pi \leftarrow$ random permutation of data points in $\mathcal{D}$
    parfor $\mathbf{x}_q \in \mathcal{D}_{\text{val}}$ ▷ Iterate over validation samples in parallel
      Sort $\mathbf{N}_q$ according to the positions in π
      $\mathbf{h} \leftarrow$ min-heap of capacity k ▷ Initialize min-heap for k neighbors
      Initialize pre-aggregate σ_q
      $v_{\text{prev}} \leftarrow 0$ ▷ Initialize previous utility
      for $(\mathbf{x}_{\alpha_i}, d_{\alpha_i}) \in \mathbf{N}_q$
        if $|\mathbf{h}| < k$
          insert $(\mathbf{x}_{\alpha_i}, d_{\alpha_i})$ into $\mathbf{h}$
          Include $\mathbf{x}_{\alpha_i}$ into pre-aggregate σ_q
        else
          $(\mathbf{x}_h, d_h) \leftarrow$ data point and score of heap root
          if $d_{\alpha_i} > d_h$
            Remove $\mathbf{x}_h$ from pre-aggregate σ_q
            Include $\mathbf{x}_{\alpha_i}$ into pre-aggregate σ_q
            update heap root of $\mathbf{h}$ with $(\mathbf{x}_{\alpha_i}, d_{\alpha_i})$
        if $\mathbf{h}$ changed ▷ Set of k-nearest neighbors changed
          $v \leftarrow V_q(f_{\text{pred}}(\mathbf{x}_q, \sigma_q))$ ▷ Utility with current neighbors
          $j \leftarrow$ position of $\mathbf{x}_{\alpha_i}$ in π
          $\phi_{\pi_j} \leftarrow \phi_{\pi_j} + (v - v_{\text{prev}})$ ▷ Update Shapley value
          $v_{\text{prev}} \leftarrow v$ ▷ Update previous utility
    $t \leftarrow$ number of permutations evaluated
    $\phi \leftarrow \frac{1}{t}\phi$ ▷ Normalise by number of iterations
  return Data Shapley values $\phi = \{\phi_1, \ldots, \phi_n\}$

(e.g., a weighted sum of their sparse vector representations) and subsequently selects the items to recommend via a non-linear operation. Since the MC procedure requires us to sequentially scan the permutation of training points and investigate the impact of an additional sample (Lines 10–11 in Algorithm 6), it will repeatedly invoke f_{pred} with one new neighbor and $k-1$ neighbors that have been seen in the previous step. This makes it possible to reuse the aggregation result of the $k-1$ neighbors from the previous step. In order to achieve this, we assume that the KNN-SR model can maintain an aggregate σ_q of the current top-k neigbor set and directly compute predictions from this via $f_{pred}(\mathbf{x}_q, \sigma_q)$, which allows us to avoid repeating redundant aggregations.

5.5.3 KMC-Shapley Algorithm

Based on the outlined characteristics, we present *KMC-Shapley* in Algorithm 7. This variant of TMC-Shapley is tailored for KNN-SR models and runs several orders of

magnitude faster (as we experimentally show in Section 5.6.1).

KMC-Shapley starts with retrieving and indexing the top-M neighbors $\{\mathbf{x}_{\alpha_1}, \ldots, \mathbf{x}_{\alpha_M}\}$ of each validation sample $\mathbf{x}_q$ in Lines 4–5. The parameter M denotes the maximum number of neighbors to consider per validation sample and is typically set to a high number such as 500. This parameter is inspired by the common practice to limit the number of neighbors to consider in KNN recommendations via a similarity threshold or a time-based cut-off [52, 70, 99]. Moreover, recent statistics research showed that high-cardinality subsets (with more than 100 elements) produce unreliable estimates of the marginal contribution in DSV computations [61]. Analogous to TMC-Shapley, our algorithm runs several MC iterations, and generates a random permutation π of the data points in $\mathcal{D}$ in each iteration (Line 7). In contrast to TMC-Shapley, KMC-Shapley does not iterate through all data points in $\mathcal{D}$, but processes each validation sample $\mathbf{x}_q$ independently in parallel (Line 8).

For each validation sample $\mathbf{x}_q$, our algorithm needs to consider its indexed neighbors only, according to their order in the permutation π (Line 9), as only the addition of new neighbors can produce a non-zero marginal contribution for $\mathbf{x}_q$. Our algorithm iterates through these neighbors (Line 13) and maintains the set of top-k neighbors in a binary heap (Lines 15 and 22). It simultaneously maintains the corresponding pre-aggregate σ_q of the top-k neigbor set under changes (Lines 16, 20 and 21). The computation of the marginal contribution of adding a new neighbor $\mathbf{x}_\alpha$ is only necessary (e.g., potentially non-zero) if the top-k neighbor set for $\mathbf{x}_q$ changes. In such cases, the change in utility is computed and used to update the DSV estimate corresponding to $\mathbf{x}_\alpha$ (Lines 23–27). Finally, the Data Shapley values are normalised by the number of iterations conducted and returned (Lines 28–30).

Complexity analysis. We analyse the time complexity of Algorithm 7. Let P denote the number of permutations considered. The proposed algorithm runs P iterations over $|\mathcal{D}_{\text{val}}|$ validation samples, and has to process at most M neighbors per validation sample. Sorting these according to their positions in the permutation π can be done in $O(M \log M)$. For each neighbor, there is a cost of $\log k$ for a potential update of the heap and a constant cost of a single summation and subtraction for maintaining the pre-aggregate σ_q. Since $M \gg k$, this results in an overall time complexity of $O(P\,|\mathcal{D}_{\text{val}}|\,M \log M)$.

The proposed characteristics of KNN-SR algorithms can also be used to optimise the computation of the leave-one-out error, we refer to Section 5.9.2 for details.

Toy example. We visualise the advantages of KMC-Shapley over TMC-Shapley on a toy example in Figure 5.2. For that, we detail how both algorithms process a permutation π of a dataset set D with eight elements $\mathbf{x}_1, \ldots, \mathbf{x}_8$ for a KNN-SR model with $k = 2$ and a single validation sample $\mathbf{x}_q$. The operations shown correspond to Lines 4–10 in Algorithm 6 for TMC-Shapley and to Lines 9–24 in Algorithm 7 for KMC-Shapley.

On the left side, TMC-Shapley enumerates eight subsets of D according to the permutation π and evaluates the utility V of the resulting predictions for the validation sample $\mathbf{x}_q$ each time. KMC-Shapley (on the right side) takes the similarities of the data points in D to the validation sample $\mathbf{x}_q$ into account and directly maintains the top-2 neighbor set of $\mathbf{x}_q$ (highlighted in blue). This allows our algorithm to skip redundant computations, since only changes in the top-2 neighbor set of $\mathbf{x}_q$ will lead to changes

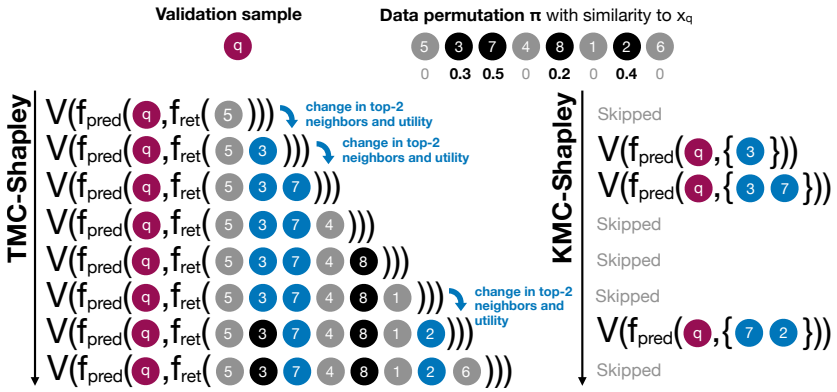


Figure 5.2: Comparison of TMC-Shapley to KMC-Shapley for a toy example with eight data points, a single validation sample $\mathbf{x}_q$ and a model with $k = 2$. TMC-Shapley (left side) processes eight data subsets for the permutation π and evaluates the utility function V each time. KMC-Shapley (on the right side) takes the similarities of the data points to $\mathbf{x}_q$ into account and directly maintains the top-2 neighbor set of $\mathbf{x}_q$ (highlighted in blue). This allows KMC-Shapley to skip many redundant computations, since only changes in the top-2 neighbor set of $\mathbf{x}_q$ lead to changes in utility.

in utility. Concretely, KMC-Shapley skips the utility computations for adding the data points $\mathbf{x}_5$, $\mathbf{x}_4$, $\mathbf{x}_1$ and $\mathbf{x}_6$ which have a similarity of zero to the validation $\mathbf{x}_q$. Moreover, KMC-Shapley can also skip the utility computation for adding $\mathbf{x}_8$, since the similarity of $\mathbf{x}_8$ to $\mathbf{x}_q$ is too small to change the top-2 neighbor set. Overall, we see that KMC-Shapley can significantly reduce the number of required utility evaluations.

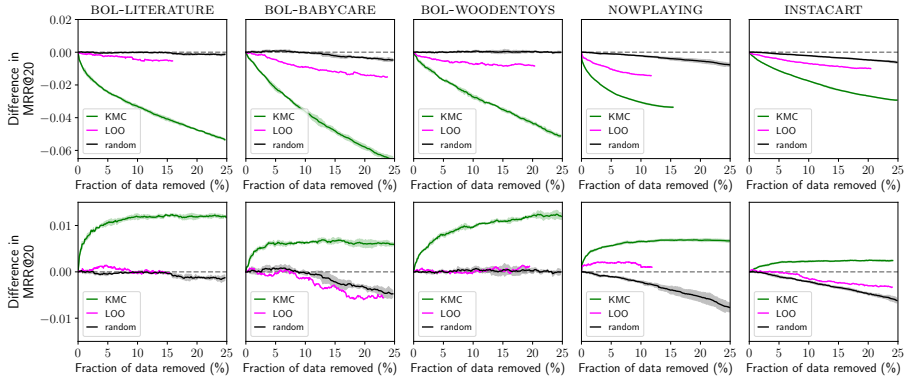
5.6 Evaluation

We evaluate the efficiency and scalability of ILLOOMINATE and experimentally validate that it identifies impactful data points.

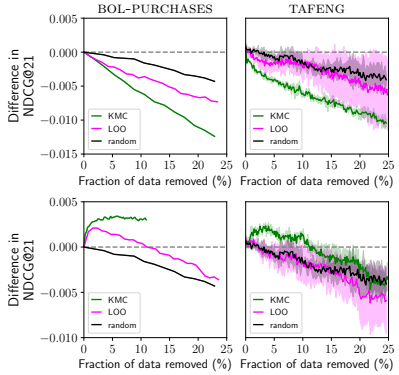
5.6.1 Efficiency

The goal of our first experiment is to showcase that our proposed optimizations for KMC-Shapley (Algorithm 7) drastically reduce the runtime compared to TMC-Shapley (Algorithm 6).

Experimental setup. We run this experiment on a Windows 10 machine with an Intel i9-10900KF CPU. We use a sample of a public session dataset with 250,000 clicks as training data and 14,058 clicks as validation data, compute DSVs for session-based recommendation, and repeat each run five times for $k \in \{50, 100, 250, 500\}$. We measure the mean runtime for a single MC iteration over all training data points with different algorithm variants. The variant `tmc` denotes a Rust implementation of the vanilla TMC-Shapley algorithm (Algorithm 6), which retrains the recommendation model each time. Next, we introduce our proposed optimizations step by step to



(a) Removing browsing sessions in SBR.

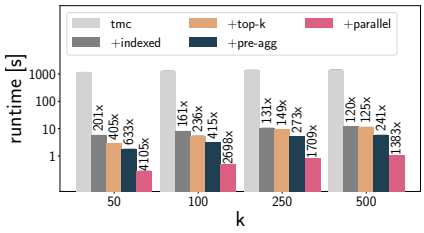


(b) Removing purchase histories in NBR.

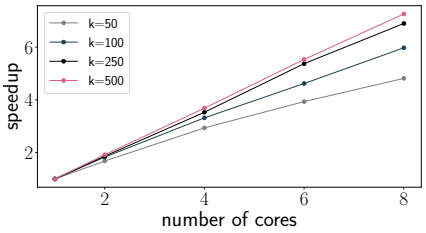
Figure 5.3: Impact of removing the most important data points (top row) and data points with negative importance scores (bottom row) from recommendation systems for different datasets. Data identified by KMC-Shapley has a stronger impact than data identified by the leave-one-out error (LOO) or randomly removed data.

transform TMC-Shapley into KMC-Shapley: `+indexed` denotes a variant that does not retrain the KNN-SR model, but reuses the indexed neighbors per validation sample instead; the `+top-k` variant additionally maintains the top-k neighbor set in a binary heap and only computes marginal contributions once this set changes; the `+pre-agg` variant additionally maintains a pre-aggregate for accelerating inference and the final optimization `+parallelism` parallelizes the computation with eight threads.

Results and discussion. We plot the resulting mean runtimes (and the speedup over the `tmc` variant) on a logarithmic scale in Figure 5.4(a). We observe that all our performance optimizations are beneficial, as each optimization further reduces the runtime. In this experiment, KMC-Shapley is three orders of magnitude faster than the vanilla TMC-Shapley implementation which repeatedly retrains the model. As expected,



(a) Microbenchmark for the benefits of our proposed optimizations. Each optimization reduces the runtime of the Data Shapley value estimation.



(b) Scalability evaluation of KMC-Shapley: its runtime scales linearly with the number of cores available, with diminishing returns for smaller values k.

Figure 5.4: Efficiency and scalability of KMC-Shapley.

the largest runtime reduction originates from reusing the neighbor sets and avoiding model retraining in the `+indexed` variant. In this experiment, we find that our algorithm can conduct a single iteration for all training data points in a second or less for all values of k.

The results confirm that it is indeed possible to design customized, highly accelerated variants of the Shapley value computation for specific classes of algorithms. Furthermore, the short runtimes shows that our algorithm can be run by data scientists on their consumer laptops, without expensive accelerator hardware.

5.6.2 Scalability

The goal of the next experiment is to show that our implementation benefits from additional computational resources, such as CPU cores, and is able to handle datasets with millions of clicks on a consumer-level laptop.

Experimental setup. We experiment with a session-based recommender on a large sample of 10,431,353 clicks in 1,668,295 sessions on Bol. We run KMC-Shapley on a validation set containing 250,000 clicks in 40,023 sessions, vary the number of neighbors k, and increase the number of cores from one to eight. We run this experiment on a machine with a Mac M1 Pro, repeat each run six times, and report the speedup over the single-threaded baseline.

Results and discussion. We plot the resulting speedups in Figure 5.4(b). We observe that the runtime scales linearly with the number of available cores, which is expected due to the map-reduce-like computational pattern in KMC-Shapley. We observe diminishing returns for smaller values of k, which we attribute to the fact that the computation is less dominated by the computational efforts for inference in these cases. Even on this large dataset of 10 million data points, a single KMC iteration with eight cores only takes 136 seconds on average for all training data points..

Table 5.1: Datasets for session-based recommendation.

Dataset		#sessions	#items	#clicks		
				train	valid	test
BOL-LITERATURE	proprietary	393,655	50,202	1,704,661	168,109	166,289
BOL-BABYCARE	proprietary	29,324	3,161	127,088	13,351	13,395
BOL-WOODENTOYS	proprietary	98,420	8,937	437,329	44,427	44,825
NOWPLAYING	public	97,922	196,531	489,367	58,277	57,616
INSTACART	public	21,068	18,661	124,376	60,104	59,363

5.6.3 Impact of Data Removal

Next, we evaluate how well ILLOOMINATE identifies impactful data points for our recommendation data. For that, we adopt a common data removal experiment [32, 50, 61] to our recommendation setup. In this experiment, data points are removed from the training data according to a given order (e.g., sorted ascendingly or descendingly with respect to their Data Shapley values), and the prediction quality of a model trained on the remaining data is repeatedly evaluated on a held-out test set. The rate at which the prediction quality changes indicates how impactful the chosen removal order is.

Session-Based Recommendation

We first conduct a data removal experiment for session-based recommendation [26, 70].

Experimental setup. We leverage the VMIS-kNN recommendation algorithm [52] with hyperparameters $k = 50$ and $M = 500$. We experiment with samples from two public datasets (NOWPLAYING, INSTACART) and three proprietary click datasets (BOL-LITERATURE, BOL-BABYCARE, BOL-WOODENTOYS) with up to 1.7 million clicks from different product categories on Bol. We prepare the datasets as follows: we conduct a temporal split of the sessions into training sessions and held-out sessions, and we randomly split the held-out sessions into 50% validation and 50% test data. We make sure that we only retain items seen in the training data (e.g., we ignore cold-start items for which no clicks have been seen yet). Table 5.1 summarises the statistics of the resulting datasets. We compute Data Shapley values via KMC-Shapley and LOO errors with respect to the MRR for the first 20 recommended items (referred to as MRR@20).

We evaluate the impact of removing highly important sessions with positive DSVs in descending order. We repeat this analogously for sessions with positive leave-one-out errors and also include a baseline, which simply removes data points at random. We replicate this experiment for removing low-scored data, where we remove the sessions with negative DSVs and LOOs in ascending order instead.

Results and discussion. In Figure 5.3(a) we plot the resulting absolute differences in MRR@20 for the removal of up to 25% of the training sessions. After repeating each data removal experiment three times, we plot the mean values as solid lines and the standard deviation as shaded areas. When removing highly important data, we observe significant differences between notions of data valuation. Removing the training sessions

Table 5.2: Datasets for next-basket recommendation.

Dataset		#users	#purchases	#baskets	#items
BOL-PURCHASES	proprietary	97,867	2,309,445	417,028	376,672
TAFENG	public	1,000	41,421	6,586	6,586

according to their Data Shapley values drastically reduces the MRR@20 scores by up to 0.06, which indicates that KMC-Shapley performs significantly better at identifying high-impact data points. Removing training sessions according to the LOO error still reduces the scores faster than random removal, but the corresponding sessions have a lower impact on MRR@20 scores, compared to removal by Data Shapley values.

For removing data with negative importance scores, we again observe the strongest positive impact with Data Shapley values. This removal even slightly improves the prediction quality in all datasets, a confirmation that negatively scored data points often correspond to corrupted data, which hurts model performance. We do not observe such consistent improvements for removal according to the LOO.

Next-Basket Recommendation

We repeat the data removal experiment for next-basket recommendation.

Experimental setup. The setup is analogous to the previous experiment with the following differences. We use the TIFU-kNN model [43], which is deployed in production on online grocery shopping platforms of brands from our partner companies [114]. We leverage NDCG@21 as an evaluation metric [8], which is common for this recommendation task. We experiment on a large proprietary dataset called BOL-PURCHASES containing a sample of more than two million purchase events from Bol. Additionally, we experiment with a sample of 1,000 users from the public dataset TAFENG. Table 5.2 presents the dataset characteristics. We use the purchase history of each user as training data, from which we hold out their most recent shopping basket. We randomly split these most recent baskets into validation and test data. Note that we remove user histories from the training data (analogous to the previous experiment) so that they will not occur in the neighbor sets, but still leverage a user's personal history for their personalized prediction [64].

Results and discussion. In Figure 5.3(b) we plot the resulting differences in NDCG@21 for removing up to 25% of the training sessions. The results observed are in line with the results from the previous experiment on session-based recommendation.

5.7 Applications

We present several ongoing applications of ILLOOMINATE at Bol. Note that all applications use the DSV for computing data importance, since it clearly outperforms the LOO error in identifying impactful data.

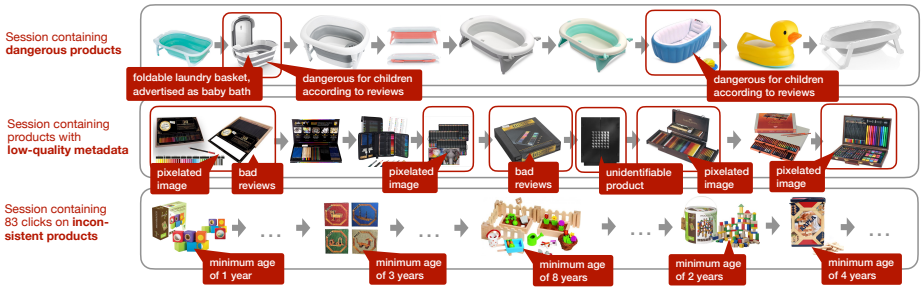


Figure 5.5: Examples of sessions with negative importance scores in Bol's click data (*each box represents a single session consisting of a sequence of clicks on items, arrows show the order of clicks*). We encounter sessions containing dangerous products (e.g., a foldable laundry basket incorrectly advertised as baby bath), sessions with metadata quality issues (e.g., pixelated and unidentifiable images) as well as sessions with inconsistent products (wooden toys for different age ranges).

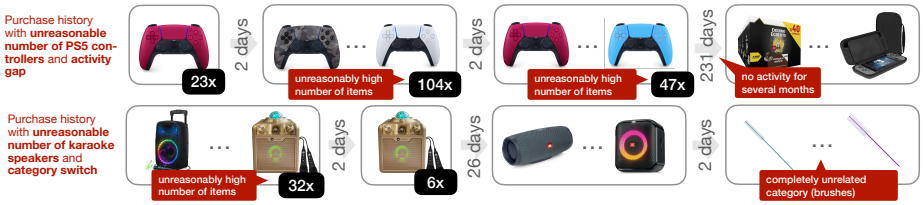


Figure 5.6: Examples of purchase histories with negative importance scores from Bol (*boxes indicate items bought together in a shopping cart, arrows point to the next shopping cart*). The histories contain electronics items bought in unreasonably high numbers (e.g., over a hundred PlayStation controllers), an activity gap, and switches to completely unrelated product categories (coffee, dental products) at some point.

5.7.1 Identifying Outliers and Corrupted Data

The main purpose of ILLOOMINATE is to help us find corrupted and harmful interaction data in our production recommendation system *Serenade* [53]. This session-based recommendation serves the "others also viewed" recommendations on our product pages. Apart from improving our recommendation systems, uncovering data issues is also important for other teams, e.g., those responsible for product quality and risk issues on the platform.

In order to showcase this use case, we apply our library to a large sample of historical click data from the babycare, wooden toys, and wooden pencils categories, and compute DSVs with MRR@20 as the target metric. We find that the lowest-scored sessions contain inconsistencies, problematic products, and corrupted data. We show three example sessions taken from the ten lowest-scored sessions found in these data samples in Figure 5.5. Note that each box in this figure represents a single session consisting of a sequence of clicks on items, and that arrows indicate the order of clicks. All of these sessions are harmful to the recommendation system (i.e., including them in the training data negatively impacts prediction quality) and suffer from data issues. The

top-most session originates from a user exploring baby bath products and for example contains a foldable laundry basket, which has been incorrectly advertised as a baby bath by its seller, and which according to its reviews is actually dangerous to use for bathing children. The remaining two sessions shown in the figure suffer from different issues. The second session contains pencil products, many of which suffer from quality issues in their metadata, such as pixelated and unidentifiable images. The third session contains the massive number of 83 clicks on wooden toys, and the toys target highly variable age ranges (one-year-olds to eight-year-olds), making this session unsuitable as a basis for recommendation.

These examples showcase that DSVs identify low-quality interaction data with issues that are hard to anticipate upfront. Such analyses typically result in a set of follow-up actions. First, low-quality interaction data should be filtered from the training data of our recommendation systems. We are currently in the process of preparing A/B tests for recommendations trained on such filtered data. Moreover, other teams (e.g., the risk team) are made aware of the uncovered product issues and will take corresponding actions, e.g., by removing dangerous products such as the foldable laundry basket from the platform.

We repeat this analysis with ILLOOMINATE for a large sample of purchase histories from Bol and a next-basket recommendation model, leveraging NDCG@20 as the target metric. We again find that the lowest-scored purchase histories contain data that is not helpful for our recommendation systems. To showcase this, we visualise two of the ten lowest-scored purchase histories in Figure 5.6. The histories contain electronics items bought in unreasonable numbers (e.g., over a hundred PlayStation controllers), switch to completely unrelated product categories (coffee, dental products) at some point, and are therefore clearly uninformative for a recommendation system. We attribute such purchase histories to commercial accounts, which buy large numbers of products for several thousands of Euros as a response to price promotions, probably intended for re-selling. Note that simply removing all interactions from commercial accounts decreases prediction quality according to our experience. Therefore, ILLOOMINATE provides us with a means to identify data with negative impact originating from commercial accounts on a fine-granular level.

5.7.2 Increasing the Sustainability of Recommendations via Data Pruning

A major goal of Bol is to make sustainable shopping[3] easier for customers. We present an experimental application of DSVs, which contributes to this mission. The aim here is to increase the number of sustainable items in the predictions of our session-based recommendation *Serenade*, without compromising the prediction quality. A model-centric approach would involve modifying the recommendation engine and forcing it to apply a higher weight to sustainable products. However, this approach might lead to lower-quality recommendations that are more speculative as opposed to being grounded in historical retrieval data. Contrary to this, we choose a data-centric approach which prunes the underlying data in an effort to remove the histories containing unsustainable

[3] https://over.bol.com/en/sustainability/

Figure 5.7: Example for the increase in sustainable products with eco-label in the recommendations for a baby care product after optimising the training set based on a utility function incorporating item sustainability.

products. This allows us maintain the prediction quality of our recommendation system by letting the real histories with a higher number of sustainable products serve as a basis for future recommendations.

In order to modify the behavior of our recommendation system through interventions on the training data, we need to choose a utility function that is able to measure the desired behavior. To this end, we augment the product metadata with a binary flag that indicates whether an item was produced in a sustainable manner. Furthermore, we modify the MRR metric to include a sustainability term that expresses the number of sustainable products in a given recommendation. Specifically, we define a utility function, which we call *SustainableMRR@t* as $0.8 \cdot MRR@t + 0.2 \cdot \frac{s}{t}$. This utility combines the MRR@t with the "sustainability coverage term" $\frac{s}{t}$, where s denotes the number of sustainable items among the t recommended items.

For this internal experiment, we use ILLOOMINATE to compute Data Shapley values and leave-one-out errors for our recommendation system with SustainableMRR@21 as a utility on click datasets from the baby care and wooden toys categories on Bol, which contain large numbers of sustainable products. Next, we prune the training data based on the computed DSVs (by removing the 5% lowest scored data points) and evaluate the recommendations on held-out test data. We observe that removing this data significantly increases the SustainableMRR@21 metric across all cases and that both the MRR@21 as well as the sustainability coverage increase as a result of the pruning. Pruning based on LOO leads to unreliable results in our experiment since we observe a small increase for the wooden toys category, but a decrease in SustainableMRR@21 for recommendations of baby care items.

In Figure 5.7, we visualise how this pruning impacts an actual recommendation: we show the top-recommended items for a session with a single click on a baby oil[4] item before and after pruning. We see that the recommendations contain two more sustainable items (care products with the "nordic ecolabel"[5]) once the training data

[4] https://www.bol.com/nl/nl/p/baby-pakket/9300000004594165/
[5] https://www.nordic-swan-ecolabel.org

Figure 5.8: Examples of high-value sessions in Bol's click data (*each box represents a single session consisting of a sequence of clicks on items, arrows show the order of clicks*). These sessions contain highly consistent product selections from related categories.

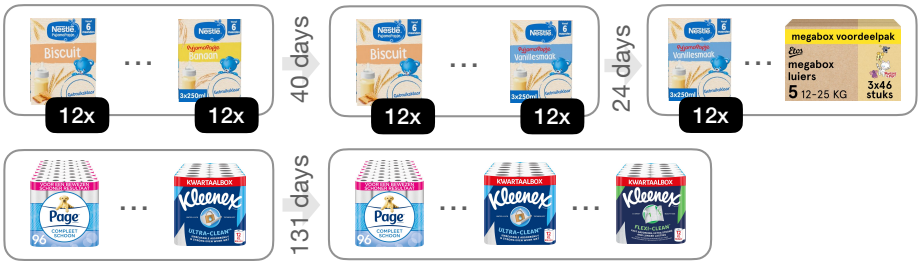


Figure 5.9: Examples of high-value purchase histories from Bol (*boxes indicate items bought together in a shopping cart, arrows point to the next shopping cart*). We encounter high turn-over items (baby formula, toilet paper) purchased in bulk in regular intervals.

has been optimised for sustainability. In summary, these results confirm that we can leverage DSVs with custom-designed utility functions for the data-centric optimisation of existing recommendation models. We are currently in the process of preparing an A/B test for the resulting recommendation.

5.7.3 High-Value Data and High-Level Insights

Apart from identifying harmful and corrupted data, we showcase that DSVs computed by ILLOOMINATE also help us understand which data is most valuable for our recommendation systems. For that, we illustrate two examples from the top ten highest scored sessions from the previous analysis (Section 5.7.1) in Figure 5.8. These sessions contain clicks on highly rated items from the wooden toys and wooden pencils categories. The sessions are highly consistent and focus on products with well-maintained metadata and high ratings. We also show two purchase histories from the top ten highest scored purchase histories for the next-basket recommendation model from the previous analysis in Figure 5.9. These purchase histories contain high turn-over items (baby formula, diapers and paper towels) purchased in bulk in regular intervals.

Apart from inspecting individual browsing sessions or purchase histories, DSVs also contribute to the high-level understanding of the interaction data in our recommendation systems. To give an example, we group the sessions from the datasets in Section 5.6.3 by their age in days and their length (the number of items), and plot these against the normalised mean Data Shapley values of the sessions in Figure 5.10. We can observe

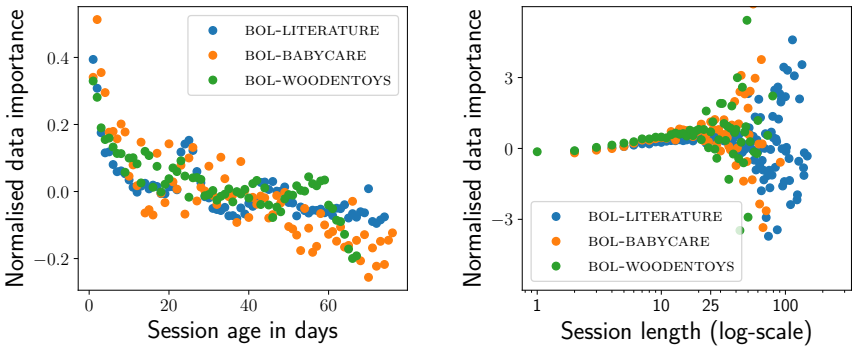


Figure 5.10: Relationship of the data importance of sessions to their age (in days) and length (in number of items) for various product categories.

that sessions from the days before the prediction time clearly are most important for the recommendation systems. Sessions that are between three weeks and eight weeks old have lower but constant importance, and the impact starts to decline afterwards. This is in line with general observations about the focus on timely data made in academic papers [70]. Moreover, we observe that the importance of sessions grows with their length, but only up to a certain point (which differs per category). This insight helps with designing features for other use cases on interaction data, e.g., detecting bots visiting the website.

5.7.4 Transferability to Neural SR Methods

Finally, we present an experimental use case, where we investigate whether the DSVs computed via our KNN-SR models also "transfer" to other models, as an indication that they identify "generally" important data. This is inspired by recent work on using DSVs from cheap KNN classifiers [46, 50] as proxies for more expensive models. We re-use the DSVs computed for the wooden toys category in Section 5.6.3 and repeat the data removal experiment presented there. However, instead of using the original VMIS-kNN model, we retrain four neural recommendation models from the recbole library [128] instead. For GRU4Rec [40] and the self-attention-based approach GC-SAN [123], we observe that the DSVs identify impactful data more reliably than the LOO error and that the results differ from random removal (see Figure 5.11). The results for the other two tested models are inconclusive. We think that these preliminary findings pose an interesting research direction to pick up by the academic community.

5.8 Conclusion

We have presented a library for the scalable debugging of interaction data for our recommendation systems at Bol, which identifies impactful sessions and purchase histories in datasets with millions of interactions and low-quality products, or improving the ecological sustainability of recommendations via data pruning.

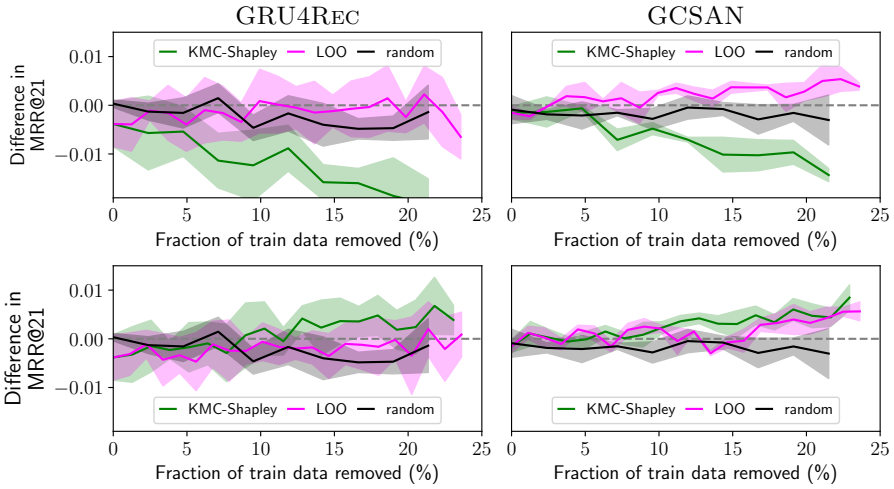


Figure 5.11: Repetition of the removal experiment for neural SBR models with importance scores computed by KMC-Shapley on a KNN-SR model. The results indicate the potential of our importance scores to generalise to other models.

In the future, we plan to include additional models [6, 24, 29] and data importance algorithms [61, 118] in our library. We also aim to investigate other directions for increasing the sustainability of our recommendations, e.g., by steering them towards products with lower weight or lower volume (to reduce logistics costs). Furthermore, we hope to inspire follow-up work on leveraging the scores from our library as proxies for expensive neural models.

This concludes the final research chapter of this thesis. In the next chapter, we discuss our main findings and future research directions.

5.9 Appendix

We provide additional details on our recommendation models in production and on the efficient LOO computation in ILLOOMINATE.

5.9.1 Recommendation Algorithms

We list the KNN-SR models in production at Bol and our partner companies, and detail how they fit our computational model outlined in Section 5.4.

Session-based recommendation. For SBR, we recently proposed the VMIS-kNN algorithm [52, 70], which is the basis for Bol's product page recommendations. In this algorithm, the interaction sequences are represented as sparse binary vectors in item space, and the retrieval function f_{ret} conducts a nearest neighbor search based on a custom, non-symmetric similarity function that computes the item overlap between a query session and the sessions in the training data, and weighs matches based on the

positions of overlapping items. The item scoring function f_{pred} aggregates the top neighbors, weighted by several custom factors such as the the position of the first item match between a neighbor and the query session and the "inverse document frequencies" of items in the session.

Next-basket recommendation. For NBR, ILLOOMINATE contains the TIFU-kNN [43] algorithm, a variant of which is also in production usage at online grocery shopping platforms in Europe [114]. TIFU conducts a two-level hierarchical aggregation of groups of shopping baskets into a sparse vector $\mathbf{x}_u$ in item space, which represents the purchase history of a user. The baskets $[\mathbf{s}_0^u, \ldots, \mathbf{s}_n^u]$ of a user u are first partitioned into groups of m elements, each of which is aggregated into a group vector $\mathbf{v}_b^u = \frac{1}{m} \sum_{i=0}^{m} r_b^{m-i} \mathbf{s}_{b+i}^u$, where b denotes the start index of the group, and r_b is a temporal decay rate to down-weight the influence of older baskets. The final representation $\mathbf{x}_u = \frac{1}{g} \sum_{i=0}^{p} r_g^{g-i} \mathbf{v}_{b+i}^u$ of a user u is computed via a subsequent aggregation of the g group vectors with another temporal decay rate r_g. The retrieval and item scoring functions are conceptually simple, in our implementation f_{ret} conducts a nearest neighbor search based on the Jaccard similarity and f_{pred} computes a linear combination $w\,\mathbf{x}_u + (1-w) \sum_{i=1}^{k} \mathbf{x}_{\alpha_i}$ of the user's own purchase vector representation $\mathbf{x}_u$ with the neighbor representations $\mathbf{x}_{\alpha_1}, \ldots, \mathbf{x}_{\alpha_k}$, where w denotes the weight put on the personal purchases.

Within-basket recommendation. This task is handled by the PerNIR model [6], which is also in production usage at online grocery shopping platforms in Europe [114]. It computes a sparse representation $(\mathbf{u}_u, \mathbf{C}_u)$ for the interaction history of each user u, where the first component $\mathbf{u}_u \in \mathbb{R}^{|I|}$ represents the personal consumption pattern of the user and the second component $\mathbf{C}_u \in \mathbb{R}^{|I| \times |I|}$ represents the item co-occurrence pattern in their shopping baskets. These components are computed via a temporally weighted aggregration of the user's past shopping baskets:

$$\mathbf{u}_u = \sum_{t=0}^{|\mathbf{h}_u|-1} \frac{1}{|\mathbf{h}_u| - t} \mathbf{s}_t$$

$$\mathbf{C}_u = \left[\sum_{t=0}^{|\mathbf{h}_u|-1} \frac{1}{|\mathbf{h}_u| - t} \frac{1_{\{i_i \in \mathbf{s}_t\}} 1_{\{i_j \in \mathbf{s}_t\}}}{|I(i_i, \mathbf{s}_t) - I(i_j, \mathbf{s}_t)|} \right]_{ij}.$$

The indexes i and j refer to items here and the function $I(i_i, \mathbf{s}_t)$ returns the position of item i_i in basket $\mathbf{s}_t$.

The retrieval function f_{ret} searches for the top-k similar users based on the cosine similarity $\mathbf{u}_q^\top \mathbf{u}_u \,/\, (|\mathbf{u}_q||\mathbf{u}_u|)$ between the personal consumption patterns $\mathbf{u}_q$ and $\mathbf{u}_u$ of the query user q and each other user u. During item scoring, the f_{pred} function of PerNIR computes a linear combination of the query user's personal vector $\mathbf{u}_q$ with the personal vectors of its neighbors as well as a linear combination of the query user's item co-occurrence matrix $\mathbf{C}_q$ with the item co-occurrence matrices of its neighbors. The resulting co-occurrence matrix is multiplied with a "selection and summation" vector built from the query user's evolving shopping basket $\mathbf{s}_{n+1}^q$ to account for the set of items already present. Note that N_q refers to the set of neighbors for q retrieved by f_{ret} and

Algorithm 8 K-LOO – Scalable leave-one-out error computation for the interaction data $\mathcal{D}$ with respect to a KNN-SR model.

function K-LOO($\mathcal{D}, \mathcal{D}_{val}, V, k$)
  $\phi = \{\phi_1, ..., \phi_n\} \leftarrow 0$ ▷ Initialise LOO values
  parfor $\mathbf{x}_q \in \mathcal{D}_{val}$ ▷ Iterate over validation samples
    $\mathbf{N}_q = \{\mathbf{x}_{\alpha_1}, \ldots, \mathbf{x}_{\alpha_{k+1}}\} \leftarrow f_{ret}(\mathbf{x}_q, \mathcal{D}, k+1)$ ▷ Retrieve neighbors
    Initialize pre-aggregate σ_q from $\{\mathbf{x}_{\alpha_1}, \ldots, \mathbf{x}_{\alpha_k}\}$
    $v_q \leftarrow V(\mathbf{x}_q, f_{pred}(\sigma_q))$ ▷ Original utility for validation sample
    for $i \in k \ldots 1$
      Include $\mathbf{x}_{\alpha_{i+1}}$ into pre-aggregate σ_q
      Remove $\mathbf{x}_{\alpha_i}$ from pre-aggregate σ_q
      $\delta_{qi} \leftarrow v_q - V(\mathbf{x}_q, f_{pred}(\mathbf{x}_q, \sigma_q))$ ▷ Change in utility
      $\phi_{\alpha_i} \leftarrow \phi_{\alpha_i} + \delta_{qi}$ ▷ Update LOO for current neighbor
  return Leave-one-out errors $\phi = \{\phi_1, \ldots, \phi_n\}$

that w and v are hyperparameters to weight the individual components of PerNIR:

$$w \left[v\, \mathbf{u}_q + \frac{1-v}{|N_q|} \sum_{u \in N_q} \frac{\mathbf{u}_q^\top \mathbf{u}_u}{|\mathbf{u}_q||\mathbf{u}_u|}\, \mathbf{u}_u \right] +$$
$$(1-w) \left[v\, \mathbf{C}_q + \frac{1-v}{|N_q|} \sum_{u \in N_q} \frac{\mathbf{u}_q^\top \mathbf{u}_u}{|\mathbf{u}_q||\mathbf{u}_u|}\, \mathbf{C}_u \right] \left[\frac{1_{\{i_i \in \mathbf{s}^q_{n+1}\}}}{|\mathbf{s}^q_{n+1}| - I(i_i, \mathbf{s}^q_{n+1})} \right].$$

5.9.2 Efficient LOO Computation

We sketch the efficient LOO computation for KNN-SR algorithms in Algorithm 8. The algorithm returns the leave-one-out errors ϕ for all n elements of the dataset $\mathcal{D}$ in a parallel loop over the validation set $\mathcal{D}_{val}$. For each validation sample $\mathbf{x}_q$, we retrieve its $k+1$ corresponding neighbors and compute the original utility v_q. Next, we loop over the training samples forming the neighbors, maintain the pre-aggregate σ_q, compute the utility difference δ_{qi} when removing $\mathbf{x}_{\alpha_i}$ from the neighbor set, and update the corresponding LOO error accordingly. The time complexity of Algorithm 8 is $O(|\mathcal{D}_{val}|\, k)$ since the algorithm processes $k+1$ neighbors for each validation sample from $\mathcal{D}_{val}$.

6 Conclusion

This thesis has focused on improving session-based recommendation systems for e-commerce platforms by addressing challenges in predictive performance, latency, and cost-efficient deployment and data quality issues. We have provided insights into the architecture of the production recommendation system at Bol and have confirmed that conceptually simple algorithms provide strong performance on our proprietary e-commerce data in Chapter 2, how we can modify, implement, and productionize a state-of-the-art recommendation system in Chapter 3, how we can assess neural network recommendation models in a scalable benchmarking framework in Chapter 4. In Chapter 5, we have uncovered various data quality issues via a data importance algorithm for sequential kNN-based recommendations that scales to datasets with millions of interactions.

In this chapter we revisit the research objectives outlined in Chapter 1 and provide an overview of our key findings related to these questions in Section 6.1. We then conclude by discussing potential avenues for future research in Section 6.2.

6.1 Summary of Findings

RQ1 Which techniques can improve both the predictive performance and user acceptance of recommendations in large-scale SBR systems?

In Chapter 2 we investigated state-of-the-art methods for session-based recommendation and found that the conceptually simple nearest-neighbor-based VS-kNN approach outperforms modern neural network-based methods on e-commerce data. We additionally found that VS-kNN is multiple orders of magnitude faster to train than neural-based methods. We evaluated the impact of reducing response latency on the acceptance of recommendations in a large scale A/B test with more than 19 million sessions. By improving serving latency, we were able to significantly improve the customer experience, leading to notable increases in business-relevant metrics.

RQ2 How can we scale VS-kNN to efficiently handle billions of interactions while maintaining low response times and adhering to production requirements in a real-world SBR system?

In Chapter 3, we proposed Vector-Multiplication-Indexed-Session kNN (VMIS-kNN), an adaption of a state-of-the-art nearest neighbor approach VS-kNN that leverages a prebuilt index to compute next-item recommendations within 1.7 milliseconds on all datasets in an offline load test. This design allows VMIS-kNN to handle hundreds of millions of historical clicks and provide recommendations at request time, even under production constraints. Building on this approach, we designed and implemented SERENADE, a scalable session-based recommendation system that is operational at Bol. We evaluated the predictive performance of VMIS-kNN, and showed that SERENADE can answer a thousand recommendation requests per second with a 90th percentile latency of less than seven milliseconds, requiring only two vCPU's, even when recommending from a catalog of millions of items. Furthermore, we presented results from a three-week-long online A/B test with up to 600 requests per second for 6.5 million distinct items on more than 45 million user sessions from our e-commerce platform. To the best of our knowledge, we provided the first empirical evidence that the superior predictive performance of nearest-neighbor approaches to session-based recommendation in offline evaluations translates to superior performance in a real-world e-commerce setting.

RQ3 How can we automatically evaluate the inference performance of SBR models under different deployment options?

In Chapter 4, we introduce ETUDE that automates the benchmarking of neural network-based SBR models in Kubernetes using synthetic click workloads and provides detailed latency and throughput metrics to guide cost-efficient deployment decisions. Our experiments reveal that GPU acceleration significantly reduces latency for large catalogs (over one million items), while smaller catalogs can be efficiently handled with CPUs. ETUDE also identified inefficiencies in the implementation of three open-source models from the Recbole library, leading to notable improvements in industry practices. End-to-end evaluations demonstrate ETUDE's ability to recommend deployment setups that balance performance and cost.

RQ4 How can we efficiently compute Data Shapley values for sequential kNN-based recommendation systems on real-world datasets with millions of datapoints?

We introduce the KMC-Shapley algorithm in Chapter 5, which leverages the sparsity of real-world user-item interaction datasets and the locality inherent in kNN-based recommendation models deployed in production. By focusing on impactful neighbors, the algorithm reduces redundant computations and efficiently evaluates utility changes. Our experimental evaluation demonstrates that KMC-Shapley significantly enhances both efficiency and scalability on both public and proprietary datasets, applied to session-based and next-basket recommendation tasks. These optimizations result in several orders of magnitude speedup over existing methods, making it feasible to

process millions of interactions. Building on this approach, we developed the ILLOOMINATE library, which enables scalable Data Shapley value computation for kNN-based recommendation systems.

RQ5 Are Data Shapley values helpful for debugging real-world interaction data in sequential kNN-based recommendation systems?

In Chapter 5, we investigate how ILLOOMINATE utilizes Data Shapley values to systematically debug public and proprietary real-world interaction data in recommendation systems. By applying these values, we identify problematic data points, including sessions containing dangerous products or items with poor-quality metadata, such as pixelated images or misleading labels. Pruning these negatively scored data points results in measurable improvements in metrics such MRR and NDCG, demonstrating the efficacy of Data Shapley values in optimizing training data. Additionally, we introduce a metric, Sustainable-MRR which aligns with Bol's sustainability objectives by prioritizing data associated with sustainable products. By identifying and pruning data that is inconsistent with these sustainability goals, we achieve not only increased MRR but also a measurable increase in the ecological sustainability of recommendations.

6.2 Future Work

In this section, we discuss the limitations of the work in this thesis and possible directions for extending it. In Chapters 2 and 3 we have explored directions to enhance the product page recommendations at Bol, where we focus on both algorithmic and systems-related aspects. However, one important factor which we did not address is how the presentation of recommendations — particularly the title accompanying them — can influence their effectiveness. From user interviews, we discovered that some customers find the title "Others Also Viewed" misleading, as it suggests that the recommendations are based on the actions of other users, while in reality, they are generated from the customer's own interactions on the website during their session. This misalignment could raise questions such as: *How does the choice of title, compared to the recommendation method, influence user engagement and the adoption of behavior-adaptive recommendations?*

In Chapter 4 we have studied evaluating inference latency for neural models for session-based recommendations. Our experiments have revealed that the scenarios with a catalog size beyond 100,000 items need multiple CPU's or accelerated hardware. An interesting research question to answer is *how the incorporation of techniques to trade-off prediction quality with inference latency, such as model quantization [31] or approximated nearest neighbor search [48], as well as the automatic choice of appropriate instance types for declarative specified workloads, improve the efficiency and scalability of session-based recommendation systems?*

In Chapter 5, we present promising findings indicating that the importance scores generated by sequential kNN-based recommendation models can potentially serve as proxies for neural network based session recommendation models, which are often too resource-intensive to retrain frequently. This opens up a valuable new direction for researchers to explore, with a central research question emerging: *How can importance*

scores from KNN-SR models serve as effective proxies for neural models? Investigating this question could lead to more computationally efficient methods for maintaining high-quality recommendations without the need for frequent neural model retraining.

Additionally, in real-world scenarios, interaction data is continuously growing as new user interactions and sessions are recorded. This raises another important question: *How can data importance be incrementally maintained when new data is added, without recalculating the importance for the same data points each time?* Follow-up work should focus on developing methods to address this challenge, ensuring that the data importance scores are updated efficiently without unnecessary recomputation of already processed data. This could significantly reduce computational overhead and thus contribute to the adoption of data importance scores in real-world sequential kNN-based production systems, making them feasible for large-scale, continuously evolving datasets.

We hope that this thesis serves as a catalyst for future research that reduces the gap between academia and industry.

Bibliography

[1] Z. Abedjan, X. Chu, D. Deng, R. C. Fernandez, I. F. Ilyas, M. Ouzzani, P. Papotti, M. Stonebraker, and N. Tang. Detecting data errors: where are we and what needs to be done? *Proc. VLDB Endow.*, 9:993–1004, Aug. 2016. ISSN 2150-8097. doi: 10.14778/2994509.2994518. URL https://doi.org/10.14778/2994509.2994518. (Cited on page 60.)

[2] Actix. Actix Web - a powerful, pragmatic, and extremely fast web framework for Rust, 2023. URL https://actix.rs. (Cited on pages 31 and 49.)

[3] Albert Heijn. Van land tot klant: onze ketens, 2023. URL https://www.ah.nl/over-ah/duurzaamheid/onze-ketens. (Cited on page 52.)

[4] X. Amatriain. Building industrial-scale real-world recommender systems. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, page 7–8, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312707. doi: 10.1145/2365952.2365958. (Cited on pages 12, 25, and 41.)

[5] I. Arapakis, X. Bai, and B. B. Cambazoglu. 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*, SIGIR '14, page 103–112, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450322577. doi: 10.1145/2600428.2609627. (Cited on pages 19, 21, 24, and 44.)

[6] M. Ariannezhad, M. Li, S. Schelter, and M. de Rijke. A personalized neighborhood-based model for within-basket recommendation in grocery shopping. In *Proceedings of the Sixteenth ACM International Conference on Web Search and Data Mining*, pages 87–95, 2023. (Cited on pages 58, 59, 62, 79, and 80.)

[7] Ars Technica. Lazy use of AI leads to Amazon products called "I cannot fulfill that request", 2024. URL https://arstechnica.com/ai/2024/01/lazy-use-of-ai-leads-to-amazon-products-called-i-cannot-fulfill-that-request/. (Cited on pages 2 and 58.)

[8] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*, volume 463. ACM press New York, 1999. (Cited on pages 62, 64, and 73.)

[9] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473. (Cited on pages 12 and 16.)

[10] J. Bai, F. Lu, K. Zhang, et al. Onnx: Open neural network exchange, 2019. URL https://github.com/onnx/onnx. (Cited on page 54.)

[11] Bol.com. Facts and figures about bol.com, 2023. URL https://pers.bol.com/en/facts-figures/. (Cited on page 52.)

[12] Buzzfeed. "Amazon's choice" does not necessarily mean a product is good, 2019. URL https://www.buzzfeednews.com/article/nicolenguyen/amazons-choice-bad-products. (Cited on page 58.)

[13] B. Chandramouli, J. J. Levandoski, A. Eldawy, and M. F. Mokbel. StreamRec: a real-time recommender system. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, page 1243–1246, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450306614. doi: 10.1145/1989323.1989465. (Cited on pages 12, 25, and 60.)

[14] Channel 4 News. Potentially deadly bomb ingredients are 'frequently bought together' on Amazon, 2017. URL https://www.channel4.com/news/potentially-deadly-bomb-ingredients-on-amazon. (Cited on pages 2 and 58.)

[15] T. Chen, H. Yin, H. Chen, R. Yan, Q. V. H. Nguyen, and X. Li. Air: Attentional intention-aware recommender systems. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 304–315, 2019. doi: 10.1109/ICDE.2019.00035. (Cited on pages 25 and 60.)

[16] K.-J. Cho, Y.-C. Lee, K. Han, J. Choi, and S.-W. Kim. No, that's not my feedback: Tv show recommendation using watchable interval. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 316–327, 2019. doi: 10.1109/ICDE.2019.00036. (Cited on pages 25 and 60.)

[17] P. Covington, J. Adams, and E. Sargin. Deep neural networks for YouTube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, page 191–198, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340359. doi: 10.1145/2959100.2959190. (Cited on page 1.)

[18] daryheap2021. d-ary heap. https://docs.rs/dary_heap/0.3.0/dary_heap/, 2021. (Cited on

page 29.)

[19] A. S. Das, M. Datar, A. Garg, and S. Rajaram. Google news personalization: scalable online collaborative filtering. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, page 271–280, New York, NY, USA, 2007. Association for Computing Machinery. ISBN 9781595936547. doi: 10.1145/1242572.1242610. (Cited on pages 12, 14, and 25.)

[20] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston, and D. Sampath. The YouTube video recommendation system. In *Proceedings of the Fourth ACM Conference on Recommender Systems*, RecSys '10, page 293–296, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589060. doi: 10.1145/1864708.1864770. (Cited on pages 1, 12, and 14.)

[21] T. Dunning and E. Friedman. *Practical Machine Learning: Innovations in Recommendation*. " O'Reilly Media, Inc.", 2014. (Cited on page 25.)

[22] Ekta and R. Pradhan. Valuation-based data acquisition to improve machine learning fairness. *Quality in Databases workshop at VLDB*, 2150:8097, 2024. (Cited on page 60.)

[23] M. Y. Eltabakh, Y. Tian, F. Özcan, R. Gemulla, A. Krettek, and J. McPherson. Cohadoop: flexible data placement and its exploitation in hadoop. *Proc. VLDB Endow.*, 4(9):575–585, June 2011. ISSN 2150-8097. doi: 10.14778/2002938.2002943. (Cited on page 31.)

[24] G. Faggioli, M. Polato, and F. Aiolli. Recency aware collaborative filtering for next basket recommendation. In *Proceedings of the 28th ACM Conference on User Modeling, Adaptation and Personalization*, UMAP '20, page 80–87, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450368612. doi: 10.1145/3340631.3394850. URL https://doi.org/10.1145/3340631.3394850. (Cited on pages 62 and 79.)

[25] X. Fan, Z. Liu, J. Lian, W. X. Zhao, X. Xie, and J.-R. Wen. Lighter and better: Low-rank decomposed self-attention networks for next-item recommendation. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, page 1733–1737, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380379. doi: 10.1145/3404835.3462978. (Cited on pages 1, 44, and 46.)

[26] M. Ferrari Dacrema, P. Cremonesi, and D. Jannach. Are we really making much progress? a worrying analysis of recent neural recommendation approaches. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, page 101–109, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362436. doi: 10.1145/3298689.3347058. URL https://doi.org/10.1145/3298689.3347058. (Cited on pages 62 and 72.)

[27] L. Flokas, W. Wu, Y. Liu, J. Wang, N. Verma, and E. Wu. Complaint-driven training data debugging at interactive speeds. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 369–383, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392495. doi: 10.1145/3514221.3517849. URL https://doi.org/10.1145/3514221.3517849. (Cited on page 60.)

[28] C. Gao, X. He, D. Gan, X. Chen, F. Feng, Y. Li, T.-S. Chua, and D. Jin. Neural multi-task recommendation from multi-behavior data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1554–1557, 2019. doi: 10.1109/ICDE.2019.00140. (Cited on pages 25 and 60.)

[29] D. Garg, P. Gupta, P. Malhotra, L. Vig, and G. Shroff. Sequence and time aware neighborhood for session-based recommendations: Stan. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR'19, page 1069–1072, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361729. doi: 10.1145/3331184.3331322. URL https://doi.org/10.1145/3331184.3331322. (Cited on pages 62 and 79.)

[30] B. G. Gebre, K. Ranta, S. van den Elzen, E. Kuiper, T. Baars, and T. Heskes. Pfeed: Generating near real-time personalized feeds using precomputed embedding similarities. *ArXiv*, abs/2402.16073, 2024. URL https://api.semanticscholar.org/CorpusID:267938115. (Cited on page 1.)

[31] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer. A survey of quantization methods for efficient neural network inference. In *Low-Power Computer Vision*, pages 291–326. Chapman and Hall/CRC, 2022. (Cited on pages 54 and 85.)

[32] A. Ghorbani and J. Zou. Data Shapley: Equitable valuation of data for machine learning. In *International Conference on Machine Learning*, pages 2242–2251, 2019. (Cited on pages 2, 4, 6, 58, 59, 60, 61, 65, and 72.)

[33] Google. General-purpose machine family for Compute Engine, 2023. URL https://cloud.google.com/compute/docs/general-purpose-machines#e2_machine_types. (Cited on page 49.)

[34] Google. Compute Engine Pricing, 2023. URL https://cloud.google.com/compute/all-pricing.

(Cited on page 54.)
[35] googleblogIntroducingTensorFlow. Introducing TensorFlow Feature Columns — developers.googleblog.com. https://developers.googleblog.com/2017/11/introducing-tensorflow-feature-columns.html, 2017. [Accessed 28-09-2023]. (Cited on page 50.)
[36] L. Guo, H. Yin, Q. Wang, B. Cui, Z. Huang, and L. Cui. Group recommendation with latent voting mechanism. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 121–132, 2020. doi: 10.1109/ICDE48307.2020.00018. (Cited on pages 25 and 60.)
[37] Z. Hammoudeh and D. Lowd. Training data influence analysis and estimation: a survey. *Mach. Learn.*, 113(5):2351–2403, Mar. 2024. ISSN 0885-6125. doi: 10.1007/s10994-023-06495-7. URL https://doi.org/10.1007/s10994-023-06495-7. (Cited on pages 58, 60, and 61.)
[38] J. He, J. Qi, and K. Ramamohanarao. A joint context-aware embedding for trip recommendations. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 292–303, 2019. doi: 10.1109/ICDE.2019.00034. (Cited on pages 25 and 60.)
[39] A. Heidari, J. McGrath, I. F. Ilyas, and T. Rekatsinas. Holodetect: Few-shot learning for error detection. In *Proceedings of the 2019 ACM SIGMOD International Conference on Management of Data (SIGMOD '19)*. ACM, 2019. URL https://api.semanticscholar.org/CorpusID:102353360. (Cited on page 60.)
[40] B. Hidasi and A. Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, CIKM '18, page 843–852, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450360142. doi: 10.1145/3269206.3271761. (Cited on pages 17, 61, and 78.)
[41] B. Hidasi, A. Karatzoglou, L. Baltrunas, and D. Tikk. Session-based recommendations with recurrent neural networks. In Y. Bengio and Y. LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.06939. (Cited on pages 12, 13, 17, 25, and 34.)
[42] Y. Hou, B. Hu, Z. Zhang, and W. X. Zhao. Core: Simple and effective session-based recommendation within consistent representation space. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '22, page 1796–1801, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450387323. doi: 10.1145/3477495.3531955. (Cited on pages 1, 44, and 46.)
[43] H. Hu, X. He, J. Gao, and Z.-L. Zhang. Modeling personalized item frequency information for next-basket recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1071–1080, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380164. doi: 10.1145/3397271.3401066. (Cited on pages 25, 34, 58, 59, 62, 64, 66, 73, and 80.)
[44] Y. Huang, B. Cui, W. Zhang, J. Jiang, and Y. Xu. Tencentrec: Real-time stream recommendation in practice. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, page 227–238, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450327589. doi: 10.1145/2723372.2742785. (Cited on pages 12, 14, 25, and 60.)
[45] D. Jannach and M. Ludewig. When recurrent neural networks meet the neighborhood for session-based recommendation. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*, RecSys '17, page 306–310, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346528. doi: 10.1145/3109859.3109872. (Cited on pages 13, 17, 25, and 62.)
[46] R. Jia, D. Dao, B. Wang, F. A. Hubis, N. M. Gurel, B. Li, C. Zhang, C. Spanos, and D. Song. Efficient task-specific data valuation for nearest neighbor algorithms. *Proc. VLDB Endow.*, 12 (11):1610–1623, July 2019. ISSN 2150-8097. doi: 10.14778/3342263.3342637. URL https://doi.org/10.14778/3342263.3342637. (Cited on pages 59, 60, 65, and 78.)
[47] R. Jia, F. Wu, X. Sun, J. Xu, D. Dao, B. Kailkhura, C. Zhang, B. Li, and D. Song. Scalability vs. utility: Do we have to sacrifice one for the other in data importance quantification? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8239–8247, 2021. doi: 10.1109/CVPR46437.2021.00814. (Cited on page 59.)
[48] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3):535–547, 2021. doi: 10.1109/TBDATA.2019.2921572. (Cited on pages 54 and 85.)
[49] W.-C. Kang and J. McAuley. Self-attentive sequential recommendation. In *2018 IEEE International Conference on Data Mining (ICDM)*, pages 197–206, Los Alamitos, CA, USA, Nov. 2018. IEEE Computer Society. doi: 10.1109/ICDM.2018.00035. (Cited on pages 44 and 46.)
[50] B. Karlaš, D. Dao, M. Interlandi, S. Schelter, W. Wu, and C. Zhang. Data debugging with Shapley importance over machine learning pipelines. In *The Twelfth International Conference on Learning*

Representations, 2023. (Cited on pages 58, 59, 60, 61, 72, and 78.)

[51] B. Kersbergen and S. Schelter. Learnings from a retail recommendation system on billions of interactions at bol.com. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2447–2452, 2021. doi: 10.1109/ICDE51399.2021.00277. (Cited on pages 7, 23, 24, 25, 34, 39, 44, and 60.)

[52] B. Kersbergen, O. Sprangers, and S. Schelter. Serenade – Low-latency session-based recommendation in e-commerce at scale. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 150–159, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392495. doi: 10.1145/3514221.3517901. (Cited on pages 8, 44, 54, 58, 59, 61, 62, 63, 64, 66, 68, 72, and 79.)

[53] B. Kersbergen, O. Sprangers, F. Kootte, S. Guha, M. de Rijke, and S. Schelter. Etude – Evaluating the inference latency of session-based recommendation models at scale. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 5177–5183, Los Alamitos, CA, USA, May 2024. IEEE Computer Society. doi: 10.1109/ICDE60146.2024.00389. (Cited on pages 8, 60, and 74.)

[54] B. Kersbergen, O. Sprangers, B. Karlaš, M. de Rijke, and S. Schelter. Illoominate – Scalable debugging of recommendation data in e-commerce. Under submission, 2025. (Cited on page 8.)

[55] P. W. Koh and P. Liang. Understanding black-box predictions via influence functions. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 1885–1894. JMLR.org, 2017. (Cited on page 60.)

[56] R. Kohavi, A. Deng, B. Frasca, T. Walker, Y. Xu, and N. Pohlmann. Online controlled experiments at large scale. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '13, page 1168–1176, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450321747. doi: 10.1145/2487575.2488217. (Cited on pages 19 and 21.)

[57] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37, 2009. doi: 10.1109/MC.2009.263. (Cited on pages 12, 24, and 25.)

[58] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017. ISSN 0001-0782. doi: 10.1145/3065386. (Cited on pages 12 and 16.)

[59] kubeservices2021. Kubernetes networking services. https://kubernetes.io/docs/concepts/services-networking/service/, 2021. (Cited on page 32.)

[60] H. W. Kuhn and A. W. Tucker. *Contributions to the Theory of Games*. Number 28 in Annals of Mathematics Studies. Princeton University Press, 1953. (Cited on page 61.)

[61] Y. Kwon and J. Y. Zou. Beta Shapley: a unified and noise-reduced data valuation framework for machine learning. In *International Conference on Artificial Intelligence and Statistics*, 2021. URL https://api.semanticscholar.org/CorpusID:239998535. (Cited on pages 4, 58, 59, 60, 61, 65, 68, 72, and 79.)

[62] J. J. Levandoski, M. Sarwat, M. F. Mokbel, and M. D. Ekstrand. RecStore: an extensible and adaptive framework for online recommender queries inside the database engine. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, page 86–96, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450307901. doi: 10.1145/2247596.2247608. (Cited on pages 12, 25, and 60.)

[63] J. Li, P. Ren, Z. Chen, Z. Ren, T. Lian, and J. Ma. Neural attentive session-based recommendation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, page 1419–1428, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450349185. doi: 10.1145/3132847.3132926. (Cited on pages 12, 13, 17, 23, 25, 34, 44, and 46.)

[64] M. Li, S. Jullien, M. Ariannezhad, and M. de Rijke. A next basket recommendation reality check. *ACM Trans. Inf. Syst.*, 41(4), Apr. 2023. ISSN 1046-8188. doi: 10.1145/3587153. (Cited on pages 25, 34, 58, 62, and 73.)

[65] P. Li, X. Rao, J. Blase, Y. Zhang, X. Chu, and C. Zhang. Cleanml: A study for evaluating the impact of data cleaning on ml classification tasks. In *Proceedings of the 2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 13–24, 2021. URL https://api.semanticscholar.org/CorpusID:229666398. (Cited on page 60.)

[66] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1):76–80, 2003. doi: 10.1109/MIC.2003.1167344. (Cited on page 58.)

[67] Q. Liu, Y. Zeng, R. Mokhosi, and H. Zhang. Stamp: Short-term attention/memory priority model for session-based recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 1831–1839, New York, NY, USA, 2018.

Association for Computing Machinery. ISBN 9781450355520. doi: 10.1145/3219819.3219950. (Cited on pages 1, 2, 12, 13, 17, 23, 25, 34, 44, and 46.)

[68] Z. Liu, Z. Zhou, and T. Rekatsinas. Picket: guarding against corrupted data in tabular data during learning and inference. *The VLDB Journal*, 31(5):927–955, Oct. 2021. ISSN 1066-8888. doi: 10.1007/s00778-021-00699-w. URL https://doi.org/10.1007/s00778-021-00699-w. (Cited on page 60.)

[69] Z.-P. Liu, L. Zou, X. Zou, C. Wang, B. Zhang, D. Tang, B. Zhu, Y. Zhu, P. Wu, K. Wang, and Y. Cheng. Monolith: Real time recommendation system with collisionless embedding table. *ArXiv*, abs/2209.07663, 2022. URL https://api.semanticscholar.org/CorpusID:252355135. (Cited on page 1.)

[70] M. Ludewig, N. Mauro, S. Latifi, and D. Jannach. Performance comparison of neural and non-neural approaches to session-based recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, RecSys '19, page 462–466, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362436. doi: 10.1145/3298689.3347041. (Cited on pages 3, 6, 12, 13, 17, 18, 19, 21, 23, 24, 25, 26, 33, 34, 44, 58, 61, 62, 68, 72, 78, and 79.)

[71] M. Ludewig, N. Mauro, S. Latifi, and D. Jannach. Empirical analysis of session-based recommendation algorithms: A comparison of neural and non-neural approaches. *User Modeling and User-Adapted Interaction*, 31(1):149–181, Mar. 2021. ISSN 0924-1868. doi: 10.1007/s11257-020-09277-1. (Cited on page 44.)

[72] X. Luo and J. Pei. Applications and computation of the Shapley value in databases and machine learning. In *Companion of the 2024 International Conference on Management of Data*, SIGMOD-/PODS '24, page 630–635, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704222. doi: 10.1145/3626246.3654680. URL https://doi.org/10.1145/3626246.3654680. (Cited on pages 59 and 60.)

[73] S. Madisetty. Event recommendation using social media. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 2106–2110, 2019. doi: 10.1109/ICDE.2019.00249. (Cited on pages 25 and 60.)

[74] S. M. McNee, J. Riedl, and J. A. Konstan. Being accurate is not enough: how accuracy metrics have hurt recommender systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '06, page 1097–1101, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595932984. doi: 10.1145/1125451.1125659. (Cited on page 1.)

[75] F. McSherry, D. G. Murray, R. Isaacs, and M. Isard. Differential dataflow. In *Sixth Biennial Conference on Innovative Data Systems Research, CIDR 2013, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*. www.cidrdb.org, 2013. URL http://cidrdb.org/cidr2013/Papers/CIDR13_Paper111.pdf. (Cited on pages 37 and 41.)

[76] W. Meng, D. Yang, and Y. Xiao. Incorporating user micro-behaviors and item knowledge into multi-task learning for session-based recommendation. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1091–1100, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380164. doi: 10.1145/3397271.3401098. (Cited on page 2.)

[77] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. J. Franklin, R. Zadeh, M. Zaharia, and A. Talwalkar. MLlib: machine learning in apache spark. *J. Mach. Learn. Res.*, 17(1):1235–1241, Jan. 2016. ISSN 1532-4435. (Cited on pages 15 and 31.)

[78] New York Times. Lawmakers press Amazon on sales of chemical used in suicides, 2022. URL https://www.nytimes.com/2022/02/04/technology/amazon-suicide-poison-preservative.html. (Cited on page 58.)

[79] NVIDIA. TensorRT, an SDK for high-performance deep learning inference, 2023. URL https://developer.nvidia.com/tensorrt. (Cited on page 54.)

[80] L. Oala, M. Maskey, L. Bat-Leah, A. Parrish, N. M. Gürel, T.-S. Kuo, Y. Liu, R. Dror, D. Brajovic, X. Yao, et al. Dmlr: Data-centric machine learning research–past, present and future. *DMLR*, 2024. (Cited on page 58.)

[81] S. Owen. *Mahout in action*, volume 10. Manning Shelter Island, NY, 2012. (Cited on page 25.)

[82] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS 2017 Workshop on Autodiff*, 2017. URL https://openreview.net/forum?id=BJJsrmfCZ. (Cited on pages 46 and 48.)

[83] A. Pfadler, H. Zhao, J. Wang, L. Wang, P. Huang, and D. L. Lee. Billion-scale recommendation with heterogeneous side information at Taobao. In *36th IEEE International Conference on Data*

Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020, pages 1667–1676. IEEE, 2020. doi: 10.1109/ICDE48307.2020.00148. (Cited on pages 25 and 60.)

[84] N. Polyzotis, M. Zinkevich, S. Roy, E. Breck, and S. Whang. Data validation for machine learning. In *Proceedings of Machine Learning and Systems*, volume 1, pages 334–347, 2019. (Cited on page 60.)

[85] R. Pradhan, J. Zhu, B. Glavic, and B. Salimi. Interpretable data-based explanations for fairness debugging. In *Proceedings of the 2022 International Conference on Management of Data*, SIGMOD '22, page 247–261, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450392495. doi: 10.1145/3514221.3517886. URL https://doi.org/10.1145/3514221.3517886. (Cited on page 60.)

[86] PyTorch. TorchServe benchmark results, 2022. URL https://github.com/pytorch/serve/tree/master/benchmarks#installation-1/. (Cited on page 50.)

[87] PyTorch. JIT Optimisation, 2023. URL https://pytorch.org/docs/stable/generated/torch.jit.optimize_for_inference.html. (Cited on page 51.)

[88] M. Raasveldt and H. Mühleisen. DuckDB: an embeddable analytical database. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 1981–1984, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450356435. doi: 10.1145/3299869.3320212. (Cited on page 37.)

[89] recbole1816. How to do online serving · issue #1816. — github.com. https://github.com/RUCAIBox/RecBole/issues/1816, 2023. [Accessed 16-10-2023]. (Cited on page 44.)

[90] recbole1855. Questions about optimization and efficiency · issue #1855. — github.com. https://github.com/RUCAIBox/RecBole/issues/1855, 2023. [Accessed 16-10-2023]. (Cited on page 44.)

[91] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C.-J. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou. MLPerf inference benchmark. In *Proceedings of the ACM/IEEE 47th Annual International Symposium on Computer Architecture*, ISCA '20, page 446–459. IEEE Press, 2020. ISBN 9781728146614. doi: 10.1109/ISCA45697.2020.00045. (Cited on page 44.)

[92] S. Redyuk, Z. Kaoudi, V. Markl, and S. Schelter. Automating data quality validation for dynamic data ingestion. In *EDBT*, pages 61–72, 2021. (Cited on page 60.)

[93] P. Ren, Z. Chen, J. Li, Z. Ren, J. Ma, and M. de Rijke. RepeatNet: a repeat aware neural recommendation machine for session-based recommendation. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33014806. (Cited on pages 1, 23, 44, and 46.)

[94] F. Ricci, L. Rokach, and B. Shapira. *Introduction to Recommender Systems Handbook*, pages 1–35. Springer US, Boston, MA, 2011. ISBN 978-0-387-85820-3. doi: 10.1007/978-0-387-85820-3_1. (Cited on pages 12, 25, and 60.)

[95] rn5l2021. Performance comparison of neural and non-neural approaches to session-based recommendation - additional information. https://rn5l.github.io/session-rec/, 2021. (Cited on page 33.)

[96] rocksdb2021. RocksDB. https://rocksdb.org, 2021. (Cited on page 32.)

[97] RUCAIBox. RecBole model issues, 2023. URL https://github.com/RUCAIBox/RecBole/issues/. Specific issues: #1894 and #1895. (Cited on pages 6 and 53.)

[98] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, page 285–295, New York, NY, USA, 2001. Association for Computing Machinery. ISBN 1581133480. doi: 10.1145/371920.372071. (Cited on pages 1, 12, 14, 24, 25, and 39.)

[99] S. Schelter, C. Boden, and V. Markl. Scalable similarity-based neighborhood methods with mapreduce. In *Proceedings of the Sixth ACM Conference on Recommender Systems*, RecSys '12, page 163–170, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450312707. doi: 10.1145/2365952.2365984. URL https://doi.org/10.1145/2365952.2365984. (Cited on pages 12, 24, 25, and 68.)

[100] S. Schelter, D. Lange, P. Schmidt, M. Celikel, F. Biessmann, and A. Grafberger. Automating large-scale data quality verification. In *Proceedings of the VLDB Endowment (PVLDB)*, volume 11, page 1781–1794. VLDB Endowment, Aug. 2018. doi: 10.14778/3229863.3229867. URL https:

//doi.org/10.14778/3229863.3229867. (Cited on page 60.)
[101] S. Schelter, U. Celebi, and T. Dunning. Efficient incremental cooccurrence analysis for item-based collaborative filtering. In *Proceedings of the 31st International Conference on Scientific and Statistical Database Management*, SSDBM '19, page 61–72, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362160. doi: 10.1145/3335783.3335784. (Cited on page 25.)
[102] S. Schelter, S. Grafberger, P. Schmidt, T. Rukat, M. Kiessling, A. Taptunov, F. Biessmann, and D. Lange. Differential data quality verification on partitioned data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1940–1945, 2019. doi: 10.1109/ICDE.2019.00210. (Cited on page 60.)
[103] S. Schelter, S. Grafberger, S. Guha, B. Karlas, and C. Zhang. Proactively screening machine learning pipelines with arguseyes. In *Companion of the 2023 International Conference on Management of Data*, SIGMOD '23, page 91–94, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450395076. doi: 10.1145/3555041.3589682. URL https://doi.org/10.1145/3555041.3589682. (Cited on pages 60 and 61.)
[104] S. Shankar, L. Fawaz, K. Gyllstrom, and A. G. Parameswaran. Moving fast with broken data. *CoRR*, abs/2303.06094, 2023. doi: 10.48550/ARXIV.2303.06094. URL https://doi.org/10.48550/arXiv.2303.06094. (Cited on page 60.)
[105] H. Shojanazeri. TorchServe performance tuning, animated drawings case-study, 2022. URL https://pytorch.org/blog/torchserve-performance-tuning/. (Cited on page 50.)
[106] sidecars2021. Istio sidecars. https://istio.io/latest/docs/reference/config/networking/sidecar/, 2021. (Cited on page 32.)
[107] J. Song, Z. Li, Z. Hu, Y. Wu, Z. Li, J. Li, and J. Gao. PoisonRec: An adaptive data poisoning framework for attacking black-box recommender systems. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 157–168, 2020. doi: 10.1109/ICDE48307.2020.00021. (Cited on pages 25 and 60.)
[108] Q. Tan, J. Zhang, J. Yao, N. Liu, J. Zhou, H. Yang, and X. Hu. Sparse-interest network for sequential recommendation. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, WSDM '21, page 598–606, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450382977. doi: 10.1145/3437963.3441811. (Cited on pages 44 and 46.)
[109] Y. K. Tan, X. Xu, and Y. Liu. Improved recurrent neural networks for session-based recommendations. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems*, DLRS 2016, page 17–22, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450347952. doi: 10.1145/2988450.2988452. (Cited on pages 44 and 46.)
[110] J. Tang and K. Wang. Personalized top-n sequential recommendation via convolutional sequence embedding. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, page 565–573, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355810. doi: 10.1145/3159652.3159656. URL https://doi.org/10.1145/3159652.3159656. (Cited on page 2.)
[111] tch-rs. Rust bindings for the C++ api of PyTorch., 2023. URL https://github.com/LaurentMazare/tch-rs. (Cited on page 49.)
[112] TorchServe. TorchServe - A performant, flexible and easy to use tool for serving PyTorch models in production., 2023. URL https://pytorch.org/serve/. (Cited on page 48.)
[113] M. Tsagkias, T. H. King, S. Kallumadi, V. Murdock, and M. de Rijke. Challenges and research opportunities in ecommerce search and recommendations. *SIGIR Forum*, 54(1), Feb. 2021. ISSN 0163-5840. doi: 10.1145/3451964.3451966. (Cited on pages 25 and 60.)
[114] M. Vechtomova et al. Databricks AI summit: Streamlining API deploy ML models across multiple brands: Ahold Delhaize's experience on serverless, 2023. URL https://www.youtube.com/watch?v=GSJFyoBiCXk. (Cited on pages 59, 62, 63, 73, and 80.)
[115] Vice News. AI-generated books of nonsense are all over amazon's bestseller lists, 2023. URL https://www.vice.com/en/article/ai-generated-books-of-nonsense-are-all-over-amazons-bestseller-lists/. (Cited on page 58.)
[116] vsknnimpl2021. VS-kNN reference implementation. https://github.com/rn5l/session-rec/blob/master/algorithms/knn/vsknn.py, 2021. (Cited on page 37.)
[117] P. Walsh. batched-fn - a Rust server plugin for deploying deep learning models with batched prediction, 2023. URL https://github.com/epwalsh/batched-fn. (Cited on page 49.)
[118] J. T. Wang and R. Jia. Data Banzhaf: A robust data valuation framework for machine learning. In F. Ruiz, J. Dy, and J.-W. van de Meent, editors, *Proceedings of The 26th International Conference on Artificial Intelligence and Statistics*, volume 206 of *Proceedings of Machine Learning Research*, pages

6388–6421. PMLR, 25–27 Apr 2023. URL https://proceedings.mlr.press/v206/wang23e.html. (Cited on pages 58, 60, 61, and 79.)

[119] C.-M. Wong, F. Feng, W. Zhang, C.-M. Vong, H. Chen, Y. Zhang, P. He, H. Chen, K. Zhao, and H. Chen. Improving conversational recommender system by pretraining billion-scale knowledge graph. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2607–2612, 2021. doi: 10.1109/ICDE51399.2021.00291. (Cited on pages 25 and 60.)

[120] S. Wu, Y. Tang, Y. Zhu, L. Wang, X. Xie, and T. Tan. Session-based recommendation with graph neural networks. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.3301346. (Cited on pages 44 and 46.)

[121] W. Wu, L. Flokas, E. Wu, and J. Wang. Complaint-driven training data debugging for query 2.0. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 1317–1334, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367356. doi: 10.1145/3318464.3389696. URL https://doi.org/10.1145/3318464.3389696. (Cited on page 60.)

[122] X. Xie, F. Sun, X. Yang, Z. Yang, J. Gao, W. Ou, and B. Cui. Explore user neighborhood for real-time e-commerce recommendation. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2464–2475, 2021. doi: 10.1109/ICDE51399.2021.00279. (Cited on pages 25 and 60.)

[123] C. Xu, P. Zhao, Y. Liu, V. S. Sheng, J. Xu, F. Zhuang, J. Fang, and X. Zhou. Graph contextualized self-attention network for session-based recommendation. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, page 3940–3946. AAAI Press, 2019. ISBN 9780999241141. (Cited on pages 1, 44, 46, 61, and 78.)

[124] H. Yin, Q. Wang, K. Zheng, Z. Li, J. Yang, and X. Zhou. Social influence-based group representation learning for group recommendation. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 566–577, 2019. doi: 10.1109/ICDE.2019.00057. (Cited on pages 25 and 60.)

[125] F. Yuan, A. Karatzoglou, I. Arapakis, J. M. Jose, and X. He. A simple convolutional generative network for next item recommendation. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, WSDM '19, page 582–590, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359405. doi: 10.1145/3289600.3290975. (Cited on pages 1, 12, 13, 17, and 25.)

[126] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, page 2, USA, 2012. USENIX Association. (Cited on pages 13 and 15.)

[127] Zalando. Extending the life of fashion, 2023. URL https://corporate.zalando.com/en/our-impact/extending-life-fashion. (Cited on page 52.)

[128] W. X. Zhao, Y. Hou, X. Pan, C. Yang, Z. Zhang, Z. Lin, J. Zhang, S. Bian, J. Tang, W. Sun, et al. RecBole 2.0: Towards a more up-to-date recommendation library. In *CIKM*, pages 4722–4726, 2022. (Cited on pages 44, 46, 48, 52, and 78.)

[129] Y. Zheng, C. Gao, X. He, Y. Li, and D. Jin. Price-aware recommendation with graph convolutional networks. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 133–144. IEEE, 2020. (Cited on pages 25 and 60.)

[130] C. Zhou, J. Ma, J. Zhang, J. Zhou, and H. Yang. Contrastive learning for debiased candidate generation in large-scale recommender systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 3985–3995, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383325. doi: 10.1145/3447548.3467102. (Cited on page 1.)

[131] X. Zhou, D. Qin, X. Lu, L. Chen, and Y. Zhang. Online social media recommendation over streams. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 938–949, 2019. doi: 10.1109/ICDE.2019.00088. (Cited on pages 25 and 60.)

[132] Z. Zolaktaf, R. Babanezhad, and R. Pottinger. A generic top-n recommendation framework for trading-off accuracy, novelty, and coverage. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 149–160, 2018. doi: 10.1109/ICDE.2018.00023. (Cited on pages 25 and 60.)

Summary

This thesis provides a comprehensive exploration of scalable solutions for session-based recommendation systems, tackling key challenges, which are: large product catalogs, millions of users, sparse interaction data, and the debugging of large-scale recommendation datasets. Conducted in collaboration with Bol, a European e-commerce platform, this research bridges the gap between academic advancements and the practical demands of industrial systems.

Chapter 2 focuses on designing and deploying algorithms and frameworks that efficiently process billions of interactions with high efficiency and low latency. These algorithms and frameworks are rigorously evaluated through both offline experiments and real-world deployments.

In Chapter 3, the thesis introduces VMIS-kNN, an improved version of the state-of-the-art VS-kNN algorithm. With a more efficient time complexity and small optimizations such as a prebuilt index, VMIS-kNN enhances scalability and responsiveness, enabling recommendation computations within milliseconds. Empirical evaluations across six datasets and multiple programming language implementations demonstrate its effectiveness. Additionally, the thesis presents Serenade, a production-ready, state-ful recommendation system with high throughput and low latency. Serenade integrates seamlessly into large-scale e-commerce platforms, significantly improving user engagement business metrics.

Chapter 4 highlights the Etude framework, which provides a systematic approach for benchmarking the inference performance of neural network-based session recommendation models under various deployment scenarios.

Furthermore, in Chapter 5 introduces KMC-Shapley, a scalable method for estimating Data Shapley Values in sequential kNN-based recommendation systems. This technique enhances the debugging of large-scale recommendation datasets by combining algorithmic rigor with practical utility. The research underscores the importance of balancing predictive performance, system efficiency, and ecological sustainability.

The findings confirm the effectiveness of nearest neighbor methods in specific e-commerce contexts, the crucial impact of system latency on user acceptance, and the value of data valuation techniques in maintaining the integrity of kNN-based recommendation systems. The open-source tools and methodologies developed in this thesis advance the state-of-the-art while offering practical insights for industry professionals. By combining theoretical innovation with real-world applicability, this research makes a valuable contribution to the field of session-based recommendation systems.

Samenvatting

Deze thesis biedt een grondige verkenning van schaalbare oplossingen voor sessiegebaseerde aanbevelingssystemen en richt zich op belangrijke uitdagingen, namelijk grote productcatalogi, miljoenen gebruikers, schaarse interactiedata en het *debuggen* van grootschalige aanbevelingsdatasets. Het onderzoek, uitgevoerd in samenwerking met Bol, een Europees e-commerceplatform, slaat een brug tussen academische vooruitgang en industriële omgevingen.

Hoofdstuk 2 richt zich op de ontwikkeling en implementatie van algoritmen en frameworks die efficiënt miljarden interacties kunnen verwerken met hoge doorvoersnelheden en weinig vertraging. De effectiviteit van deze algoritmen en frameworks wordt aangetoond via zowel offline evaluaties als toepassingen in de praktijk.

In Hoofdstuk 3 wordt VMIS-kNN geïntroduceerd, een verbeterde versie van het bestaande VS-kNN-algoritme. Door gebruik te maken van een betere tijd complexiteit en kleine optimalisaties zoals een vooraf opgebouwde index, biedt VMIS-kNN betere schaalbaarheid en snellere aanbevelingen binnen milliseconden. De voordelen worden empirisch aangetoond op zes datasets en in verschillende programmeertalen. Daarnaast introduceert dit hoofdstuk Serenade, een productierijp, stateful aanbevelingssysteem met hoge doorvoersnelheid en lage latency, dat succesvol integreert in grootschalige e-commerceplatforms en een meetbare verbetering laat zien in gebruikersacceptatie en bedrijfsresultaten van Bol.

Hoofdstuk 4, introduceert het Etude-framework, dat een systematische aanpak biedt voor het benchmarken van de prestaties van neurale netwerkmodellen voor sessiegebaseerde aanbevelingen in diverse deployementscenario's.

In Hoofdstuk 5 wordt KMC-Shapley gepresenteerd, een schaalbare methode voor het schatten van Data Shapley Values binnen sequentiële kNN-gebaseerde aanbevelingssystemen. Dit draagt bij aan het debuggen van grootschalige aanbevelingsdatasets door een combinatie van theoretische precisie en praktische toepasbaarheid. Het onderzoek benadrukt daarbij het belang van een evenwicht tussen voorspellende prestaties, systeemefficiëntie en ecologische duurzaamheid.

De resultaten van deze thesis bevestigen de effectiviteit van nearest neighbor-methoden in specifieke e-commercecontexten, benadrukken het belang van systeemlatentie voor gebruikersacceptatie en laten zien hoe datavaluatie kan bijdragen aan de betrouwbaarheid van kNN-gebaseerde aanbevelingssystemen. De ontwikkelde open-source tools en methodologieën leveren niet alleen een waardevolle bijdrage aan het vakgebied, maar bieden ook bruikbare inzichten voor professionals in de praktijk.