Advancing Neural Click Models for Unbiased Learning-to-Rank

Philipp Hager

Advancing Neural Click Models for Unbiased Learning-to-Rank

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 woensdag 29 april 2026, te 13:00 uur

door

Philipp Konstantin Hager

geboren te Düsseldorf

Promotiecommissie

Promotor:	prof. dr. M. de Rijke	Universiteit van Amsterdam
Co-promotor:	dr. O. Zoeter	Booking.com
Overige leden:	dr. R. Jagerman	Google DeepMind
	prof. dr. T. Joachims	Cornell University
	prof. dr. E. Kanoulas	Universiteit van Amsterdam
	dr. S. Magliacane	Universiteit van Amsterdam
	prof. dr. C.G.M. Snoek	Universiteit van Amsterdam

Faculteit der Natuurwetenschappen, Wiskunde en Informatica

The research was carried out at the Information Retrieval Lab at the University of Amsterdam and funded by Booking.com through the Mercury Machine Learning Lab.

Cover by Philipp Konstantin Hager
Printed by Proefschriftspecialist, Zaandam

ISBN: 978-94-93539-05-1

To my father for gifting me the joy of building things,
and to my mother for the ability to complete them.

Acknowledgments

I don't think of myself as someone who always dreamed of becoming a computer scientist. I wanted to be a firefighter, then an astronaut, then a marching band drummer, a race car mechanic, a biologist, and an audio engineer — and the list is far from complete. If I'm honest, I mostly just wanted to build LEGO for a living. That I ended up here, all these years later, is the result of a chain of very happy accidents and, more than anything else, the support and encouragement of many wonderful people along the way. While any attempt to thank everyone who shaped this journey is bound to fall short, the following is mine.

First, I want to thank my two supervisors. Maarten, thank you, truly, for everything. To mention one of many memories: Early in my PhD, I felt overwhelmed reading papers I could not imagine ever writing myself. I remember you telling me, very calmly, *on your path here you collected skills and a view of the world that are uniquely yours. Now, find a way to be useful to the research community.* This sentiment, surely expressed less dramatically than remembered here, lifted a lot of pressure I had put on myself and made me feel like I belonged. Thank you for being a great mentor in work and life.

And thank you, Onno, for being my co-supervisor. When I began my PhD, our weekly discussions often felt like me trying to convince you of the prevailing problems and solutions in our field. Mostly, I did not. If anything, the opposite happened, and over the last four years, I have come to see the world much more from your point of view. Without you, I would have written a very different thesis. Thank you for your patience, your research integrity, and your strong but always well-founded opinions.

I want to thank my committee: Dr. Rolf Jagerman, prof. dr. Thorsten Joachims, prof. dr. Evangelos Kanoulas, dr. Sara Magliacane, and prof. dr. Cees G. M. Snoek. I am honoured to have you on my committee, and I am grateful for the time and thought you invested in this work.

Next, I want to thank my Paranymphs. Pooya, you have more hobbies than anyone I have ever met. Thank you for introducing a few into my life. From attending classical concerts in t-shirts, developing opinions about indie films, picking up a new instrument, and starting to run, your friendship has kept me sane throughout these years.

And thank you, Clemencia. Your friendship was a constant reminder to set aside time for a life beyond work. My success in setting boundaries is perhaps best exemplified by occupying the morning meeting slot before you, me living across the city, and you across the street. Thank you for sharing this journey with me until the end.

I want to thank my friends and colleagues from IRLab for making the last four years a tremendous experience. Shashank, from giving tutorials to near-empty rooms on the other side of the world, to bringing more sandwiches to a bike trip than we'd ever biked off, and our daily coffee walks, your friendship helped me through the more difficult and absurd corners of life. Thank you for your kindness, your vulnerability, and for being a wonderful friend.

Romain, I don't know where this thesis would be without you. Thank you for being an early friend and believer in a niche research direction. Your creativity and speed were the driving force behind a major part of this journey. Thank you for carrying me when it mattered. May we one day go skiing together on the day of our paper deadline!

To Maria, for always looking out for the people around you, despite being one of

the busiest people I know. Thank you for being a great reminder not to let family take a backseat to academia. Thank you for always being there for me, Cafe Six gang forever.

Before meeting you, Maurits, I would not have believed a living soul could convince me to work out in Westerpark before breakfast (or that it was possible to present 70 slides in 12 minutes). Thank you for listening to me vent about academia and life, and for your advice throughout the years. To my favorite *burning bridge*.

Sam, when I first saved your contact, your status read *life is soup, I am fork*. For someone who nourishes everyone around him with kindness, humor, and wisdom, I think you're at least a spoon (a ladle, even). Thank you for motivating me to make music again, for making the best pizza I had in this town, and for being a great friend.

Sami, *Schatzi*, from heated German culinary debates in my first job interview to our adventures around Taiwan, thank you for bringing so much joy to these past years. And thank you for being that little voice in my head advocating the exquisite (and typically expensive) things in life. *Did I really need that new coffee grinder? Yes. Yes, I did.*

Gabriel, I think we met properly for the first time to go stand-up paddling, and you immediately told me to start saying *yes* to more things in life. You were right. While I'll likely never get a haircut in swimming trunks in a bridal hair salon in Taiwan again, thank you for inspiring the most fun adventures and for being a wonderful human being.

Thilina, you discovering your conference presentation slot the day before and absolutely nailing your talk the next morning is still one of the most outrageous things I have witnessed. Thank you for your wit, for creating community in the lab week after week, and for never, under any circumstances, losing an argument.

Thank you, Antonis, for being an absolute delight to be around. Thong, for your kindness and for being a superb first office mate. Clara, for taking vacation with the seriousness it deserves. David, for your unwavering optimism that brightens the lab. Mariya, for your genuine interest in people. Mohammad, for our shared suffering in the classroom, and all the award-worthy conference celebrations. Jasmin, for keeping the lab running behind the scenes. Roxana, for all the thoughtful conversations. Ruben for always being the first to help and for starting the IRLab band with Vera and Syrenna. Jin, for your patient attempt to teach me how to pronounce names correctly. Panagiotis, for being an inspiring supervisor. And Amy, Chuan, Gabrielle, Jingwei, Kidist, Sid, and Zihan for enduring me as an office mate over the years.

Thank you to all other members of IRLab who made this journey an absolute privilege: Ali, Amin, Ana, Andrew, Arezoo, Barrie, Cosimo, Dan, Daniel, Dylan, Evangelos, Georgios, Hongyi, Jia-Hong, Jiahuan, Jingfen, Julien, Lu, Maarten, Maartje, Maik, Maryam, Maxime, Ming, Mohanna, Mozhdeh, Olivier, Ruqing, Shaojie, Siddharth, Simon, Teng, Tom, Vaishali, Weijia, Xinyi, Yangjun, Yibin, Yifei, Yixing, Yongkang, Yougang, Yuanna, Yubao, Yuyue, Zahra. I'd also like to thank Ivana, Petra, Pablo, and Kiki for supporting the lab throughout these years.

My gratitude extends to many other academics beyond IRLab who shaped my time as a PhD student. First of all, thank you, Harrie. You are indeed a kind of *academic uncle* to me. I have learned so much from you about writing, presenting, and conducting oneself as a researcher. Thank you for your advice, from my very first attempt at a paper until now, for fostering community, and for the finest restaurant recommendations.

Sanne, I think when we first met, you asked *do you always complain this much?* While that hasn't really changed, I've always appreciated your Dutch honesty. Thank

you for your moral compass, for seeing the bigger picture, for bullying me into running a half-marathon, and for being a friend since my very first conference.

Bram, my first memory of you is standing on stage, making an entire conference room get up and throw their hands in the air to *re-energize* before your presentation. While it was clear from the start that you are a deeply funny person, you are also an immensely caring one. Thank you for always being there for me over these years.

Norman, I'm honestly in awe of the passion you apply to everything in life and in research. Mathijs, for every joyfully heated dinner debate. Olivier, thank you for being one of the kindest people I have met in academia. Jan Malte and Ben, for always looking out for me and for being seriously great researchers without taking yourselves too seriously. Santiago, thank you for being an audience of one and for taking me salsa dancing, which made me feel, without a doubt, like the whitest person in the room. Andrew, while Scottish may get the better of me at times, I always know that whatever I missed was a razor-sharp research observation or very funny, usually it's both. Jonas, I'm glad to have met you despite our different research fields, as time spent with you is always time well spent.

Thank you, Sebastian, for hosting me at Northeastern University. Your enthusiasm, theoretical thoroughness, and openness to novel problems are truly inspiring. Thank you for your kindness and humor in our every interaction.

And most importantly, thank you, Pantelis. Without meeting you in Berlin and your encouragement and help throughout the years, I would likely never have found my way back to academia. Thank you for seeing what I could not see at the time.

I want to thank the Mercury Machine Learning Lab. Philip, thank you for your thoughtful feedback and patient explanations that had a major impact on this thesis. Pedro, for being a fellow motorsport enthusiast and a great neighbor. Davide and Oussama, thank you for all the joy you've brought to the group. Onno, without whom none of this would have happened. And Leihao, Sourbh, Stephan, Jinke, Frans, Matthijs, and Joris, for fostering a curious research environment and always engaging in deep discussions to help each other out despite our diverse topics. And lastly, thank you, Maryam, for keeping us organized all these years (a heroic effort, by any measure).

I want to thank all the members of the Ph.D. council of the Informatics Institute, especially Shruti and Frederike. Without your dedication, this council would not have persisted. A special thanks also to Sharvaree, Rob, Riccardo, and Carlo, who joined the tireless fight for better internship policies, and to Paola, for listening to our concerns.

Pooya introduced me to a wonderful group of film enthusiasts who brightened up many evenings. Thank you especially to James, Maurice, and Sarai, for welcoming me so warmly into the group.

Before my Ph.D., I met a bunch of wonderful people in my time working in Berlin who shaped this journey. First and foremost, to Meri-Kris. Thank you for being one of the most creative and free-spirited people I know, for all the meetings discussing lucid dreams rather than recommender systems, for convincing me that a toga is appropriate Estonian party attire, and for being the kind of friend who shows up when it matters most. Giorgia, for our shared love of venting, not least about our names being perpetually misspelled. Melisa, for all the support and the funniest Amazon package I've ever received. And to Leandro and David, for being great mentors and teaching me proper software engineering.

Am Ende dieser Reise gilt mein größter Dank meiner Familie und meinen Freunden zu Hause. Danke Mama, für deine Stärke, deine Fürsorge, und deinen unermüdlichen Einsatz, die diesen Weg erst ermöglicht haben. Danke, Chrissi und Caro, dass ich mich immer auf euren Beistand verlassen kann, gerade weil ich außerhalb der Universität wohl der Lebensunfähigste von uns Dreien bin. An Claus und Philipp, für den Humor und das Leben, das ihr in unsere Familie gebracht habt. An Christina, dafür dass du meine Neugier schon immer früh gefördert hast. An Odilia und Gerhard, für euer aufrichtiges Interesse an meinem Leben, von einem unvergesslichen Wochenende am Nürburgring bis heute. Und an Emil, Lotta und Valentina, denen diese Arbeit herzlich egal ist und mich immer wieder daran erinnern, was wirklich zählt.

An die Menschen, mit denen ich diesen Moment gerne geteilt hätte, und die ich in Gedanken bei mir trage. Papa, *no, you're never too old to Rock 'n' Roll if you're too young to die*. An Oma Hedwig und Opa Heinrich, an Oma Gisela und Opa Klaus.

An meine besten Freunde, Maxi und Lili, die mich seit meiner Kindheit begleiten, zum Lachen bringen, inspirieren, und immer erden, wenn es nötig ist. Danke, dass ihr immer da seid, egal wo das Leben uns hinverschlägt. An Anderson, Alex, Ariane, Chris, Chrissi, Janina, Johannes, Jonas, Jules, Lili's Alex, Malte, Matias, Kathrin, Robert, Svea, Tabea, und den gesamten Gymnastikverband 2001 e.V., ich wüsste nicht was ich ohne Euch machen würde.

An Ella, für unzählige Gespräche an Küchentischen bis tief in die Nacht. An Chrissi und Leon für eure Gastfreundschaft. An Robert, der mir die Freude am Segeln gezeigt hat. An Calvin und Lenka für jedes wundervolle Wochenende in Wien. An meine Studienstiftungsfreunde Georg, Jannik, und Käthe. Und an alle, die die unvergesslichen Abenteuer mit meinen ehemaligen Bands, den Betrayers of Babylon und dem Berlin Boom Orchestra, möglich gemacht haben, sowie an meine Musiklehrer Rita und Detlef.

To Shiangling and Steven, thank you for welcoming me into your home and for letting me write parts of this thesis at your kitchen table.

And lastly, to Catherine. I'm beyond grateful that we found each other on this journey. Thank you for making any place feel like home, and for the laughter, calmness, strength, and the many adventures you brought into my life. To quote a singing Austrian naval officer, *You brought music back into the house. I had forgotten.*

Philipp
Amsterdam
February, 2026

Contents

1 Introduction

Do you remember the last time a search engine explicitly asked if you found what you were looking for? You may remember leaving a *thumbs up* on a movie or a *star rating* on a recently purchased product in a web shop. But, arguably, most of our interactions with modern information systems are *implicit* rather than *explicit* [4, 62, 134]: we may, for instance, skip a song we do not like [189], spend less time watching a video with too many ads [107], or click on an article that looks promising, only to find we have fallen for a catchy headline [198].

In this thesis, we examine one of the most common forms of implicit feedback [62]: *clicks*, meaning instances where users select a result from a presented ranking. Specifically, we study the use of clicks in web search and recommendation. Search engines respond to search queries by sifting through billions of documents to typically deliver a result page with ten relevant documents. Recommender systems, by contrast, usually do not use an explicit query but rather a user's context and history to deliver a list of items. Both search engines and recommender systems face the same core challenge: ranking millions of items to surface those that are most relevant to a user's current information need. Performing this operation with machine learning is the task of learning-to-rank [119].

Historically, learning-to-rank, especially in web search, has been a supervised learning task in which human experts judge the relevance of each result for a given information need (e.g., on a five-point scale from "not" to "perfectly" relevant) [24, 41, 145, 209]. Ranking models are then trained to reproduce these judgements. However, expert annotations face significant limitations: they are expensive and time-consuming to collect at scale [24, 119], may be inappropriate for sensitive domains like personal email search [183], and have difficulty capturing subjective and evolving preferences in personalized domains such as music or travel [84, 142].

These limitations have motivated research on implicit feedback, particularly clicks, as an alternative supervision signal. Clicks are cheap to collect at scale, timely, and originate from actual users [96]. However, they introduce new challenges: clicks are a noisy and biased signal of user preference. Noise arises as users do not always click all relevant items or mistakenly click non-relevant ones [32]. While noise can diminish with more data [139], clicks often suffer from statistical biases that persist regardless of scale: users can only click on the subset of documents displayed to them, leading to item selection bias [137]. Even among displayed documents, users tend to pay more

attention to top-ranked items, making lower-ranked relevant documents less likely to be examined and clicked, a phenomenon known as position bias [39, 67, 153]. Users may also trust search engines to rank relevant items first and click top results regardless of actual relevance [2, 97, 98, 176], be influenced by the context of a document such as the quality [98] or appearance of surrounding results [160], or not click relevant documents when snippets already answer their questions [114, 125]. The study and mitigation of these systematic biases gave rise to the field of *unbiased learning-to-ran*k [6, 76, 139], which can be split into two major branches: *click modeling* [37] and *counterfactual learning-to-rank* [99, 183].

Click models emerged early in the web search community, with researchers aiming to explicitly model the generative process leading to a click [39, 153]. Click models explicitly model often unobserved (latent) factors such as document attractiveness, position examination, and user satisfaction as parameters in a probabilistic model which are typically estimated using expectation maximization [37]. Early examples include the position-based model (PBM) [153], which assumes that a user clicks on a document only if they examine its position and find the document relevant; the cascade model, which assumes that clicks depend on the relevance of previous items [39]; and the dynamic Bayesian network, which considers user satisfaction after clicking a document [25].

Counterfactual learning-to-rank (CLTR) emerged more recently by applying causal inference techniques, particularly inverse-propensity scoring (IPS), to learning-to-rank [99, 183]. The key idea of CLTR is to decouple bias and relevance estimation: first, biases in click data are estimated (e.g., the propensity of examining each position according to the PBM), then clicks are re-weighted during optimization to correct for these biases [3, 60, 184]. Thereby, clicks on rarely-examined items at lower positions receive more weight during model training than clicks on well-examined top positions. CLTR methods brought three advantages over click models [99]: (i) While click models traditionally estimated separate relevance parameters per query-document pair, requiring many repeated observations across positions, CLTR methods generalized over feature representations. (ii) Bias is estimated once and may be reused across different ranking models. And (iii) building on the learning-to-rank field [119], CLTR methods typically use ranking loss functions that optimize the order of documents rather than estimating relevance for each document independently, as is common in click models.

In recent years, the distinction between click modeling and CLTR has blurred. Modern iterations of click models have emerged that integrate neural networks and document features to generalize to unseen documents, a capability previously characteristic of CLTR methods. We term these approaches *neural click models*. A prominent example is the two-tower model, a neural network architecture based on the position-based model that has gained popularity for bias correction in industry settings [72, 85, 105, 195, 202]. These models are appealing for several reasons: (i) CLTR methods require practitioners to select an IPS correction matching the user behavior in a given dataset [99, 137, 175, 176], a choice typically informed by training and comparing click models. Since practitioners must train click models anyway, improving their ranking performance is a worthwhile goal. (ii) Click models naturally extend to more complex user behavior and can incorporate additional feedback signals, such as skips and dwell time [27, 203, 205]. (iii) Applications such as advertising and hotel auctions require calibrated click probabilities, which click models are designed to produce but

are often not guaranteed by learning-to-rank methods [11]. These motivations make neural click models an attractive research direction. Yet, key questions remain: what are their theoretical properties, how do they connect to propensity-based methods, and can more complex click models than the PBM be effectively translated into the neural paradigm?

Answering these questions rigorously is complicated by a broader methodological gap in unbiased learning-to-rank: the reliance on semi-synthetic simulations. Due to privacy and business concerns, search engine companies rarely release real, large-scale click datasets. Consequently, academic research in the field commonly relies on simulation experiments [5, 48, 75, 99, 137, 140, 176, 208]. While an essential tool for exploring and verifying methods, we risk creating feedback loops when models are optimized for the assumptions of a simulator rather than the complexity and messiness of real human behavior.

In this thesis, we aim to advance our understanding of neural click models for unbiased learning-to-rank. First, we investigate their theoretical foundations and relationship to propensity-based counterfactual learning-to-rank methods. We scrutinize their empirical performance and extend the two-tower paradigm to more sophisticated click models, creating highly flexible and scalable methods for bias correction. Second, we move beyond semi-synthetic simulation by conducting a comprehensive evaluation of unbiased learning-to-rank methods for position bias correction on the largest publicly available search engine dataset, assessing both neural click models and counterfactual techniques on real-world click data.

1.1 Research Outline and Questions

Counterfactual learning-to-rank using inverse-propensity scoring (IPS) and click models are two popular approaches for mitigating position bias in click data [135]. IPS-based methods mitigate position bias by reweighting clicks inversely to the probability of a user observing a document. Click models, in contrast, jointly infer position bias and document relevance as latent parameters. IPS approaches were initially introduced to improve on click models by requiring fewer observations per query-document pair, using features rather than separate parameters per document, decoupling bias and relevance estimation into separate steps, and optimizing pairwise or listwise ranking objectives [99, 137, 183]. However, recent neural click models, particularly the two-tower models used in industry, also leverage feature inputs and could incorporate pre-estimated position bias [195]. This convergence naturally leads us to our first research question:

RQ1 Are neural click models and inverse-propensity scoring equivalent approaches for addressing position-bias in a pointwise ranking setting?

To address this question, Chapter 2 compares a simple pointwise IPS method [13, 157] with its click model counterpart, a neural implementation of the position-based model (PBM), on equal footing. We show theoretically that when position bias is known, a standard CLTR assumption [1, 99, 137, 157, 176], both approaches optimize for unbiased document relevance. While we expected equivalent empirical ranking

performance, our simulation experiments reveal that IPS slightly outperforms the neural click model on two of three datasets. Further investigation reveals that this gap arises because the click model's gradients are biased toward documents with higher examination probabilities when features are shared across documents with conflicting relevance, while IPS weights all positions equally.

In Chapter 3, we move beyond semi-synthetic simulation experiments, prevalent in unbiased learning-to-rank (ULTR), and conduct a reproducibility study of six established bias correction methods following the PBM on the largest available real-world click dataset released by the Chinese search engine Baidu, the Baidu-ULTR dataset [209]. In this chapter, we ask:

RQ2 Do unbiased learning-to-rank methods that work well in simulation actually improve ranking performance on the largest real-world search dataset?

Despite compelling evidence of position bias in the Baidu-ULTR dataset, established PBM-based ULTR methods fail to consistently improve ranking performance relative to naive, non-debiasing baselines. While we find that bias correction leads to robust improvements in predicting user clicks, these gains do not always translate to ranking performance measured on expert annotations. Instead, the choice of ranking loss (e.g., pointwise or listwise) and input features has a more substantial impact on ranking performance, underscoring the importance of fairly comparing bias-correction methods with their naive counterparts. This chapter shows that applying simple position bias corrections as black-box approaches is not guaranteed to improve ranking performance. In some cases, as we demonstrate with transformer-based rankers, performance can even deteriorate. Our findings call for adjusting semi-synthetic experiments to better reflect real-world challenges. Lastly, unlike in Chapter 2, we find that two neural click models, including a two-tower model, reliably outperform pointwise IPS on this benchmark.

In the next chapter, we turn our attention to understanding additive two-tower models, a prevalent type of neural click model used in industry settings due to their simplicity and flexibility to include bias features beyond position [72, 195, 202]. Two-tower models use separate neural networks (or towers) to predict user examination and document relevance, combining both towers to predict clicks. Recently, however, a puzzling observation was made in related work: the ranking performance of two-tower models decreases when trained on clicks collected by a strong logging policy, meaning a production search engine that already places relevant documents at the top. Two explanations have been proposed. The first uses the lens of causal inference and confounding [122, 200]: logging policies introduce correlations between position and relevance that two-tower models might not be able to disentangle. The second concerns identifiability [30]: the question of whether model parameters can be uniquely recovered from observational data. If a logging policy never shows the same document in different positions, infinitely many combinations of bias and relevance parameters may explain the observed clicks equally well, leading to identifiability problems [30, 135]. Chapter 4 investigates two-tower models more deeply, asking:

RQ3 When can two-tower models reliably disentangle bias and relevance signals from clicks, and what role do the production systems play collecting the clicks?

To answer this question, we conduct a theoretical analysis of two-tower model identifiability and logging policy effects on parameter estimation. We prove that two-tower models can be identified not only through randomized document swaps (as previously shown) but also through overlapping document feature distributions across ranks, sometimes eliminating the strict need for randomized online experiments. Contrary to previous work, we demonstrate that logging policies have no effect on well-specified models (where model assumptions match user behavior), and find that the observations in previous work are mostly an artifact of an improper simulation setup. However, we do find that logging policies may amplify bias in misspecified models when prediction errors correlate with document placement, which we demonstrate in three realistic scenarios. Finally, we propose a propensity weighting scheme to reduce logging policy effects when model misspecification cannot be avoided.

Two-tower models are a neural parameterization of the PBM, using separate neural networks to predict user examination and document relevance. This approach gained popularity for its flexibility in incorporating bias features beyond position and its compatibility with modern deep learning frameworks. However, the PBM makes a strong assumption: clicking on one document is independent of all other documents in a ranking. More sophisticated click models, such as the user browsing model [56] or the dynamic Bayesian network [25], capture richer user behaviors, such as sequential browsing and user satisfaction post-click. These models are traditionally formulated as probabilistic graphical models (PGMs) with latent variables and optimized using expectation maximization (EM), which iteratively alternates between imputing expected values for the latent variables and updating the model parameters. While EM is a powerful tool, its iterative nature scales poorly to large datasets and requires model-specific derivations [102, 127, 171]. In Chapter 5, we therefore investigate if we can systematically apply the idea of neural parameterization and gradient-based optimization in two-tower models to a broader class of PGM-based click models:

RQ4 Can gradient-based optimization replace Expectation-Maximization to enable scalable and flexible neural parameterization of PGM-based click models?

We begin Chapter 5 by discussing the theoretical connection between EM and gradient-based optimization for click models, showing that both methods optimize the same marginal-likelihood objective and establishing gradient-based optimization as a valid alternative. Building on this insight, we introduce CLAX, a JAX-based framework that replaces EM with direct gradient-based optimization for traditional click models. We empirically show that CLAX achieves comparable click prediction performance to a prevalent EM-based click model library while training orders of magnitude faster. CLAX's modular design enables the integration of custom deep learning components into classic click models and implements embedding compression techniques from the deep recommender systems literature for scaling to large datasets. We demonstrate this scalability by experimenting on the full Baidu-ULTR dataset, comprising over 1B user sessions, on a single GPU. Finally, we evaluate CLAX models as ranking models on expert annotations and show that the neural parameterizations of more sophisticated click models beyond the PBM can surpass established two-tower models in ranking performance, suggesting practical benefits from considering neural click models with richer user behavior assumptions.

1.2 Main Contributions

In the following, we list the main contributions of this thesis.

1.2.1 Algorithmic contributions

- A propensity weighting scheme to correct for unequal logging policy exposure when training misspecified two-tower models (Chapter 4).
- A formulation of ten click models for numerically stable gradient-based optimization with log probabilities (Chapter 5).
- A method for learning mixture distributions over click models that enables ranking and click prediction on unseen documents (Chapter 5).

1.2.2 Theoretical contributions

- A theoretical analysis showing that a PBM click model, similar to inverse-propensity scoring, optimizes for unbiased document relevance when position bias in a given dataset is known (Chapter 2).
- A formal analysis of identifiability conditions for two-tower models, showing that either document swaps or overlapping feature distributions across ranks are required to recover model parameters from clicks (Chapter 4).
- An analysis of logging policy effects on two-tower models beyond identifiability, showing they do not affect well-specified models but may amplify bias under model misspecification (Chapter 4).
- A discussion of expectation maximization and gradient-based optimization for parameter estimation in neural click models, showing when both approaches are equivalent (Chapter 5).

1.2.3 Empirical contributions

- An empirical comparison of neural click models and IPS using semi-synthetic simulations shows that both methods have identical pointwise ranking performance when allocating separate parameters per query-document pair (one-hot encoding), but finds small but significant differences when generalizing over shared query-document features. A follow-up analysis shows that click models and IPS treat conflicting document features differently (Chapter 2).
- We reproduce six prominent ULTR methods on the largest real-world search engine dataset, Baidu-ULTR [209]. We find that ULTR methods robustly improve click prediction but do not consistently improve ranking performance (measured against expert annotations). We find that the choice of ranking loss and query-document features has a substantially larger impact on ranking performance than position bias correction techniques (Chapter 3).

- We apply position bias correction when training six transformer-based MonoBERT rankers on Baidu-ULTR from scratch with listwise and ULTR-based loss functions. Our study indicates that applying ULTR methods early during training can negatively affect transformer-based rankers (Chapter 3).
- We validate our theory on the identifiability of two-tower models in simulations, showing when two-tower models can be identified from overlapping feature distributions and when they require document swaps (Chapter 4).
- We demonstrate that logging policies do not impact well-specified two-tower models but amplify bias in misspecified models when prediction errors correlate with document placement, and validate a sample weighting technique that reduces this amplification (Chapter 4).
- We evaluate gradient-based optimization in CLAX against the most widely established expectation-maximization baseline, matching click prediction performance while training orders of magnitude faster (Chapter 5).
- We evaluate two embedding compression techniques for neural click models and use them to scale experiments to the full Baidu-ULTR dataset containing over 1.2 billion user sessions (Chapter 5).
- We evaluate the ranking performance of ten neural click models on Baidu-ULTR, showing that complex PGM-based models can outperform simple two-tower models commonly used in industry (Chapter 5).

1.2.4 Resource contributions

- We publish the PyTorch code for comparing IPS and neural click models in semi-synthetic simulation (Chapter 2).
- We publish JAX implementations of six established ULTR methods, three LTR baselines, and code for three offline position bias estimators (Chapter 3).
- We release the code and weights for six MonoBERT ranking models trained with different loss functions and ULTR bias-correction methods in JAX (Chapter 3).
- We release a pre-processed subset of the Baidu-ULTR dataset to make it more accessible to the ULTR community. We provide pre-computed query-document features from the original Baidu MonoBERT model, our MonoBERT, and traditional LTR features including BM25, TF-IDF, and Query Likelihood (Chapter 3).
- We release all JAX code and data for our simulation experiments on the identifiability and logging policy effects of two-tower models (Chapter 4).
- We publish CLAX, an open-source JAX-based framework for fast, modular gradient-based optimization of neural click models. The framework's design enables flexible model parameterization and scaling to billions of query-document pairs (Chapter 5).

1.3 Thesis Overview

This section provides an overview and guidance on how to navigate this thesis. This thesis contains six chapters, with the present chapter being the first.

The following four chapters explore the research questions outlined in Section 1.3. Chapters 2, 4, and 5 cover neural click modeling, while Chapter 3 presents a reproducibility study of the broader unbiased learning-to-rank field. We recommend reading Chapter 2 before Chapter 4, as the latter provides a deeper theoretical explanation for an empirical phenomenon observed in Chapter 2. Similarly, readers may benefit from reading the reproducibility study in Chapter 3 before Chapter 5, since it introduces a dataset on which the CLAX framework in Chapter 5 is evaluated. Finally, Chapter 6 summarizes the findings of this thesis and discusses avenues for future work.

1.4 Origins

Below, we list the publications that are the origins of each chapter.

Chapter 2 is based on the following paper:

- P. Hager, M. de Rijke, and O. Zoeter. Contrasting neural click models and pointwise IPS rankers. In *Advances in Information Retrieval: 45th European Conference on Information Retrieval*, 2023.

  PH: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing, Project administration. MdR: Funding Acquisition, Supervision, Writing – Review & Editing. OZ: Funding Acquisition, Supervision, Writing – Review & Editing.

Chapter 3 is based on the following paper:

- P. Hager, R. Deffayet, J.-M. Renders, O. Zoeter, and M. de Rijke. Unbiased learning to rank meets reality: Lessons from Baidu's large-scale search dataset. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024.

  PH and RD share first authorship. PH: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing, Project administration. RD: Data Curation, Formal Analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing. JMR: Funding Acquisition, Supervision, Writing – Review & Editing. OZ: Funding Acquisition, Supervision, Writing – Review & Editing. MdR: Funding Acquisition, Supervision, Writing – Review & Editing.

Chapter 4 is based on the following paper:

- P. Hager, O. Zoeter, and M. de Rijke. Unidentified and confounded? Understanding two-tower models for unbiased learning to rank. In *Proceedings of the 2025 International ACM SIGIR Conference on Innovative Concepts and Theories in Information Retrieval*, 2025.

  PH: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing, Project administration. OZ: Funding Acquisition, Supervision, Methodology, Validation, Writing – Review & Editing. MdR: Funding Acquisition, Supervision, Writing – Review & Editing.

Chapter 5 is based on the following paper:

- P. Hager, O. Zoeter, and M. de Rijke. CLAX: Fast and flexible neural click models in JAX. *arXiv preprint arXiv:2511.03620*, 2025.

  PH: Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Resources, Software, Validation, Visualization, Writing – Original Draft Preparation, Writing – Review & Editing, Project administration. OZ: Funding Acquisition, Supervision, Writing – Review & Editing. MdR: Funding Acquisition, Supervision, Writing – Review & Editing.

The writing of the thesis also benefited from work on the following publications:

- P. P. Analytis and P. Hager. Collaborative filtering algorithms are prone to mainstream-taste bias. In *Proceedings of the 17th ACM Conference on Recommender Systems*, 2023.
- R. Deffayet, P. Hager, J.-M. Renders, and M. de Rijke. An offline metric for the debiasedness of click models. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2023.
- S. Gupta, P. Hager, J. Huang, A. Vardasbi, and H. Oosterhuis. Recent advances in the foundations and applications of unbiased learning to rank. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2023.
- S. Gupta, P. Hager, J. Huang, A. Vardasbi, and H. Oosterhuis. Unbiased learning to rank: On recent advances and practical applications. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, 2024.
- M. de Haan and P. Hager. Understanding the effects of the Baidu-ULTR logging policy on two-tower models. In *The CONSEQUENCES'24 workshop, co-located with ACM RecSys'24*, 2024.
- P. Hager, M. de Rijke, and O. Zoeter. Are neural click models pointwise IPS rankers? In *The CONSEQUENCES'22 workshop, co-located with ACM RecSys'22*, CONSEQUENCES'22, 2022.

- P. Hager, M. de Rijke, and O. Zoeter. Unidentified and confounded? Understanding two-tower models for unbiased learning to rank (extended abstract). In *The CONSEQUENCES'25 workshop, co-located with ACM RecSys'25*, 2025.

2

Are Neural Click Models Pointwise IPS Rankers?

Users tend to pay more attention to and click more often on documents displayed at the top of a ranking, leading to position bias in click feedback [39, 67, 97]. Two primary approaches exist for mitigating this type of bias: Counterfactual learning-to-rank (CLTR) methods use inverse propensity scoring to re-weight clicks inversely to the probability of a user observing the clicked document [99, 175, 176]. Click models, in contrast, explicitly model the generative process of a click by jointly inferring position bias and document relevance as latent parameters [37].

Traditionally, these approaches differed in key ways [99]: CLTR methods generalize over document features and optimize pairwise or listwise ranking objectives, while click models infer separate relevance parameters per document and optimize pointwise objectives. In this context, *pointwise* refers to a ranking method that estimates a relevance score for each document independently, rather than a score relative to other documents. However, recent neural click models blur these distinctions by also generalizing over document features using neural networks [72, 184, 195], and pointwise CLTR variants have emerged that address the same pointwise ranking setting as click models [13, 157]. Under identical assumptions about user behavior, such as the position-based model [39, 153], the distinction between the approaches becomes unclear, leading us to ask:

RQ1 Are neural click models and inverse-propensity scoring equivalent approaches for addressing position-bias in a pointwise ranking setting?

In this chapter, we aim to answer this question by theoretically comparing a pointwise IPS estimator with a neural click model implementing the user assumptions of the position-based model. Afterwards, we perform an empirical comparison between both methods on equal footing in semi-synthetic simulation experiments.

This chapter was published as P. Hager, M. de Rijke, and O. Zoeter. Contrasting neural click models and pointwise IPS rankers. In *Advances in Information Retrieval: 45th European Conference on Information Retrieval*, 2023.

2.1 Introduction

Learning-to-rank a set of items based on their features is a crucial part of many real-world search [24, 85, 145, 164] and recommender systems [38, 66, 202]. Traditional supervised learning-to-rank uses human expert annotations to learn the optimal order of items [20, 24, 119]. However, expert annotations are expensive to collect [24] and can be misaligned with actual user preference [159]. Instead, the field of unbiased learning-to-rank seeks to optimize ranking models from implicit user feedback, such as clicks [1, 99, 137, 183, 184]. One well-known problem when learning from click data is that the position at which an item is displayed affects how likely a user is to see and interact with it [39, 97, 99, 177, 184]. Click modeling [37, 39, 56, 70, 153] and inverse-propensity scoring (IPS) [1, 90, 99, 138, 176] are two popular methods for learning rankers from position-biased user feedback. IPS-based counterfactual learning-to-rank methods mitigate position bias by re-weighting clicks during training inversely to the probability of a user observing the clicked item [99, 183]. In contrast, click models are generative models that represent position bias and item relevance as latent parameters to directly predict biased user behavior [37, 39, 56, 70, 153].

IPS approaches were introduced to improve over click models [99, 183] by: (i) requiring less observations of the same query-document pair by representing items using features instead of inferring a separate relevance parameter for each document [1, 99, 183, 184]; (ii) decoupling bias and relevance estimations into separate steps since the joint parameter inference in click models can fail [3, 135, 184]; and (iii) optimizing the order of documents through pairwise [90, 99] and listwise loss [137] functions instead of inferring independent pointwise relevance estimations for each document [99, 184].

At the same time, neural successors of click models have been introduced [17, 35, 72, 85, 195, 202] that can leverage feature inputs, similarly to IPS-based rankers. Moreover, pointwise IPS methods have been presented that address the same ranking setting as click models [13, 157]. In this chapter, we ask if both approaches are two sides of the same coin when it comes to pointwise learning-to-rank?

To address this question, we first introduce both approaches (Sections 2.2, 2.3) and show theoretically that both methods are equivalent when the position bias is known (Section 2.4). We then compare both approaches empirically on the prevalent semi-synthetic benchmarking setup in unbiased learning-to-rank (Section 2.5) and find small but significant differences in ranking performance (Section 2.6.1). We conclude by investigating the found differences by performing additional experiments (Section 2.6.2) and hypothesize that neural click models might be affected by position bias when learning from shared, sometimes conflicting, document features.

The main contributions of this chapter are:

- A theoretical analysis showing that a PBM click model optimizes for unbiased document relevance when the position bias is known.
- An empirical evaluation of both methods on three large semi-synthetic click datasets revealing small but significant differences in ranking performance.

- An analysis of the empirical differences that hint at neural click models being affected by position bias when generalizing over conflicting document features instead of treating each document separately.

2.2 Related Work

We provide an overview of probabilistic and neural click models, IPS-based counterfactual learning-to-rank, and comparisons between the two methodologies.

2.2.1 Click models

Probabilistic click models emerged for predicting user interactions in web search [37, 39, 153]. Factors that impact a user's click decision, such as an item's probability to be seen or its relevance are explicitly modeled as random variables, which are jointly inferred using maximum likelihood estimation on large click logs [37]. An early but prevailing model is the position-based model (PBM), which assumes that a click on a given item only depends on its position and relevance [39, 153]. Another prominent approach, the cascade model, assumes that users scan items from top to bottom and click on the first relevant item, not examining the documents below [39]. Follow-up work extends these approaches to more complex click behavior [25, 56, 70, 181], more elaborate user interfaces [193, 194], and feedback beyond clicks [52]. We refer to Chuklin et al. [37] for an overview.

Recent click models use complex neural architectures to model non-sequential browsing behavior [206] and user preference across sessions [28, 117]. Additionally, exact identifiers of items are typically replaced by more expressive feature representations [72, 195, 206]. In contrast to ever more complicated click models, neural implementations of the classic PBM recently gained popularity in industry applications [85, 195, 202]. So-called two-tower models input bias and relevance-related features into two separate networks and combine the output to predict user clicks [72, 195]. We use a neural PBM implementation similar to current two-tower models in this chapter and our findings on click model bias might be relevant to this community.

2.2.2 Counterfactual learning-to-rank

Joachims et al. introduced the concept of counterfactual learning-to-rank [99], relating to previous work by Wang et al. [183]. This line of work assumes a probabilistic model of user behavior, usually the PBM [90, 99, 137, 157] or cascade click model [175], and uses inverse-propensity scoring to mitigate the estimated bias from click data. The first work by Joachims et al. [99] introduced an unbiased version of the pairwise RankSVM method, Hu et al. [90] introduced a modified pairwise LambdaMART, and Oosterhuis and de Rijke suggested an IPS-correction for the listwise LambdaLoss framework [138]. Given that click models are pointwise rankers [184], we use a pointwise IPS method introduced by Bekker et al. [13] and later Saito et al. [157].

2.2.3 Comparing click models and IPS

Lastly, we discuss related work comparing IPS and click models. To our knowledge, Wang et al. [184] conduct the only experiment that compares both approaches on a single proprietary dataset. Their RegressionEM approach extends a probabilistic PBM using logistic regression to predict document relevance from item features instead of inferring separate relevance parameters per document. While the main motivation behind their work is to obtain better position bias estimates to train a pairwise IPS model, the authors also report the ranking performance of the inferred logistic regression model which can be seen as a component of a single-layer neural click model. The authors find that the click model improves rankings over a baseline not correcting for position bias, but is outperformed by a pairwise IPS approach [184, Table 4]. The authors also include two pointwise IPS approximations which are less effective than the click model and also fail to outperform the biased baseline model. Therefore, it is unclear how current pointwise methods suggested by Bekker et al. [13] and Saito et al. [157] would compare. We compare a recent pointwise IPS method with a neural PBM implementation and report experiments on three public LTR dataset unifying model architecture, hyperparameter tuning, and position bias estimation to avoid confounding factors.

Lastly, recent theoretical work by Oosterhuis [135] compares click models and IPS and their limits for unbiased learning-to-rank. Their work finds that IPS-based methods can only correct for biases that are an affine transformation of item relevance. For click models jointly inferring both relevance and bias parameters, they find no robust theoretical guarantees of unbiasedness and find settings in which even an infinite amount of clicks will not lead to inferring the true model parameters. We will discuss this work in more detail in Section 2.4 and extend their analysis to show that a click model only inferring item relevance should be in-fact unbiased.

2.3 Background

We introduce our assumptions on how position bias affects users, the neural click model, and IPS approach that we compare in this work.

2.3.1 A model of position bias

We begin by assuming a model of how position bias affects the click behavior of users. For this work, we resort to the prevalent model in unbiased learning-to-rank, the position-based model (PBM) [39, 153]. Let $P(Y = 1 \mid d, q)$ be the probability of a document d being relevant to a given search query q and $P(O = 1 \mid k)$ the probability of observing a document at rank $k \in K, K = \{1, 2, \ldots\}$; then we assume that clicks occur only on items that were observed and relevant:

$$\begin{aligned} P(C = 1 \mid d, q, k) &= P(O = 1 \mid k) \cdot P(Y = 1 \mid d, q) \\ c_{d,k} &= o_k \cdot y_d. \end{aligned} \tag{2.1}$$

For brevity, we use the short notation above for the rest of the chapter and drop the subscript q in all of our formulas assuming that the document relevance y_d is always conditioned on the current query context.

2.3.2 A neural position-based click model

A neural click model directly mirrors the PBM user model introduced in the previous section in its architecture [17, 35, 72, 195]. We use a neural network g to estimate document relevance $\hat{y}_d$ from features x_d and estimate position bias $\hat{o}_k$ using a single parameter per rank denoted by $f(k)$. We use sigmoid activations and multiply the resulting probabilities:

$$\begin{aligned} \hat{c}_{d,k} &= \sigma(f(k)) \cdot \sigma(g(x_d)) \\ \hat{c}_{d,k} &= \hat{o}_k \cdot \hat{y}_d. \end{aligned} \tag{2.2}$$

A common choice to fit neural click models is the binary cross-entropy loss between predicted and observed clicks in the dataset [72, 85, 195, 202, 206]:

$$\mathcal{L}_{\text{pbm}}(\hat{y}, \hat{o}) = -\sum_{(d,k)\in D} c_{d,k} \cdot \log(\hat{y}_d \cdot \hat{o}_k) + (1 - c_{d,k}) \cdot \log(1 - \hat{y}_d \cdot \hat{o}_k). \tag{2.3}$$

2.3.3 A pointwise IPS model

Instead of predicting clicks, IPS directly predicts the document relevance $\hat{y}_d$ and assumes an estimation of the position bias $\hat{o}_k$ is given [99, 157]. Thus, the IPS model we assume in this work only uses the relevance network g:

$$\hat{y}_d = g(x_d). \tag{2.4}$$

Bekker et al. [13] introduce a pointwise IPS loss that minimizes the binary cross-entropy between predicted and true document relevance. Note how the PBM assumption is used to recover the unbiased document relevance by dividing clicks by the estimated position bias $\hat{o}_k$:

$$\mathcal{L}_{\text{ips}}(\hat{y}, \hat{o}) = -\sum_{(d,k)\in D} \frac{c_{d,k}}{\hat{o}_k} \cdot \log(\hat{y}_d) + \left(1 - \frac{c_{d,k}}{\hat{o}_k}\right) \cdot \log(1 - \hat{y}_d). \tag{2.5}$$

2.4 Methods

2.4.1 Comparing unbiasedness

In this section, we compare the ability of the neural click model and pointwise IPS ranker to recover the unbiased relevance of an item under position bias. We begin by noting that in the trivial case in which there is no position bias, i.e., clicks are an unbiased indicator of relevance, both approaches are identical.

Proposition 1. *When correctly assuming that no position bias exists, i.e.,* $\forall k \in K, o_k = \hat{o}_k = 1$*, the click model and pointwise IPS method are equivalent:*

$$\mathbb{E}\left[\mathcal{L}_{ips}(\hat{y}, \hat{o})\right] = \mathbb{E}\left[\mathcal{L}_{pbm}(\hat{y}, \hat{o})\right] = -\sum_{(d,k)\in D} y_d \cdot \log(\hat{y}_d) + (1 - y_d) \cdot \log(1 - \hat{y}_d).$$

Second, both approaches also collapse to the same (biased) model in the case of not correcting for an existing position bias in the data.

Proposition 2. *When falsely assuming that no position bias exists, i.e.,* $\forall k \in K, \hat{o}_k = 1 \wedge o_k < 1$*, the click model and pointwise IPS method are equivalently biased:*

$$\mathbb{E}\left[\mathcal{L}_{ips}(\hat{y}, \hat{o})\right] = \mathbb{E}\left[\mathcal{L}_{pbm}(\hat{y}, \hat{o})\right] = -\sum_{(d,k)\in D} y_d o_k \cdot \log(\hat{y}_d) + (1 - y_d o_k) \cdot \log(1 - \hat{y}_d).$$

However, how do both approaches compare when inferring the unbiased document relevance under an existing position bias? Saito et al. [157] show that $\mathcal{L}_{\text{ips}}(\hat{y})$ is unbiased if the position bias is correctly estimated, $\forall k \in K, \hat{o}_k = o_k$ and users actually behave according to the PBM [157, Proposition 4.3]. The notion of an unbiased estimator is harder to apply to neural click models, since relevance is a parameter to be inferred. Instead of unbiasedness, Oosterhuis [135] looks into consistency of click models and shows that click models jointly estimating both bias and relevance parameters are not consistent estimators of document relevance. This means that there are rankings in which even infinite click data will not lead to the true document relevance estimate.

But what happens if click models do not have to jointly estimate bias and relevance parameters, but only item relevance? Since IPS approaches often assume access to a correctly estimated position bias [1, 99, 137, 157, 176], we investigate this idealized setting for the click model and show that initializing the model parameters $\hat{o}_k$ with the true position bias leads to an unbiased relevance estimate.

Theorem 1. *The click model is an unbiased estimator of relevance when given access to the true position bias:*

$$\mathbb{E}\left[\hat{y}_d\right] = \frac{o_k y_d}{\hat{o}_k}, \forall k \in K, \hat{o}_k = o_k. \tag{2.6}$$

Proof. We begin by taking the partial derivative of $\mathcal{L}_{\text{pbm}}$ with regard to the estimated document relevance $\hat{y}$ in our click model. Since the model factorizes, for ease of notation we will look at a single document and single observation:

$$\begin{aligned}
\frac{\partial \mathcal{L}_{\text{pbm}}}{\partial \hat{y}} &= -\left(c \cdot \frac{\partial}{\partial \hat{y}}\left[\log(\hat{o}\hat{y})\right] + (1-c) \cdot \frac{\partial}{\partial \hat{y}}\left[\log(1-\hat{o}\hat{y})\right]\right) \\
&= -\left(c \cdot \frac{\hat{o}}{\hat{o}\hat{y}} + (1-c) \cdot \frac{-\hat{o}}{1-\hat{o}\hat{y}}\right) \\
&= -\left(\frac{c}{\hat{y}} + \frac{-\hat{o} + \hat{o}c}{1-\hat{o}\hat{y}}\right) \\
&= -\frac{c - \hat{o}\hat{y}}{\hat{y}(1-\hat{o}\hat{y})}.
\end{aligned} \tag{2.7}$$

Next, we find the ideal model minimizing the loss by finding the roots of the derivative.

We note that this function is convex and any extrema found will be a minimum:

$$\begin{aligned} \frac{\partial \mathcal{L}_{\text{pbm}}}{\partial \hat{y}} &= 0 \\ -\frac{c - \hat{o}\hat{y}}{\hat{y}(1 - \hat{o}\hat{y})} &= 0 \\ \hat{y} &= \frac{c}{\hat{o}}. \end{aligned} \tag{2.8}$$

Lastly, in expectation we see that the obtained relevance estimate is the true document relevance when the estimated and true position bias are equal:

$$\begin{aligned} \mathbb{E}\left[\hat{y}\right] &= \frac{\mathbb{E}\left[c\right]}{\hat{o}} \\ \mathbb{E}\left[\hat{y}\right] &= \frac{oy}{\hat{o}}. \end{aligned} \tag{2.9}$$

□

Thus, given the correct position bias, we find that both the click model and IPS objective optimize for the unbiased document relevance, suggesting a similar performance in an idealized benchmark setup. But before covering our empirical comparison, we want to note one additional difference of both loss functions.

2.4.2 A difference in loss magnitude

We note one difference between the click model and IPS-based loss functions concerning their magnitude and relationship with position bias. While IPS-based loss functions are known to suffer from high variance due to dividing clicks by potentially small probabilities [167, 185], the neural click model seems to suffer from the opposite problem since both $y_{d,k}$ and $\hat{y}_{d,k}$ (assuming our user model is correct) are multiplied by a potentially small examination probability. Thus, independent of document relevance, items at lower positions have a click probability closer to zero, impacting the magnitude of the loss (and gradient). Figure 2.1 visualizes the loss for a single item of relevance $y_d = 0.5$ under varying degrees of position bias. While the pointwise IPS loss in expectation of infinite clicks always converges to the same distribution, the click model's loss gets smaller in magnitude with an increase in position bias. While the magnitude differs, the minimum of the loss, as shown earlier in Section 2.4.1, is still correctly positioned at 0.5. We will explore if this difference in loss magnitude might negatively impact items at lower positions in our upcoming experiments.

2.5 Experimental Setup

To compare click model and IPS-based approaches empirically, we use an evaluation setup that is prevalent in unbiased learning-to-rank [88, 93, 99, 136, 138, 140, 176, 177]. The main idea is to use real-world LTR datasets containing full expert annotations of item relevance to generate synthetic clicks according to our user model. Below, we describe the used datasets, the click generation procedure, as well as model implementation and training.

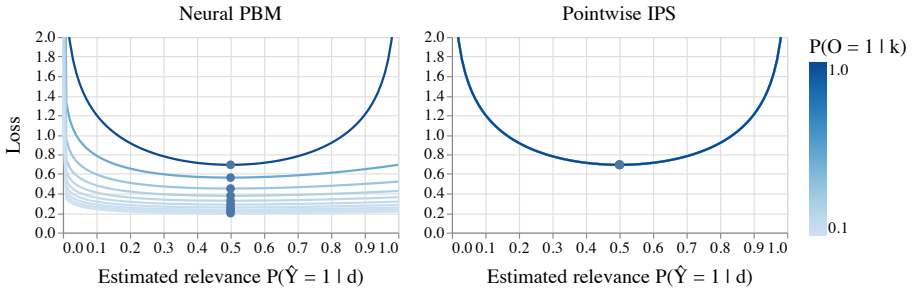


Figure 2.1: Visualizing $\mathcal{L}_{\text{pbm}}$ on the left and $\mathcal{L}_{\text{ips}}$ on the right for a single document of relevance $y_d = 0.5$ under varying degrees of position bias.

2.5.1 Datasets

We use three large-scale public LTR datasets to simulate synthetic user clicks: *MSLR-WEB30k* [145], *Istella-S* [41], and *Yahoo! Webscope* [24]. Each query-document pair is represented by a feature vector x_d and is accompanied by a score $s_d \in \{0, 1, 2, 3, 4\}$ indicating relevance as judged by a human annotator. Table 2.1 contains an overview of the dataset statistics. During preprocessing, we normalize the document feature vectors of *MSLR-WEB30k* and *Istella-S* using $\log1\text{p}(x_d) = \log_e(1 + |x_d|) \odot \text{sign}(x_d)$, as recently suggested by Qin et al. [146]. The features of *Yahoo! Webscope* come already normalized [24]. We use stratified sampling to limit each query to contain at most the 90th percentile number of documents (Table 2.1), improving computational speed while keeping the distribution of document relevance in the datasets almost identical.

2.5.2 Simulating user behavior

Our click simulation setup closely follows [176, 177]. First, we train a LightGBM [101] implementation of LambdaMART [20] on 20 sampled training queries with fully supervised relevance annotations as our production ranker.[1] The intuition is to simulate initial rankings that are better than random but leave room for further improvement.

We generate up to 100 million clicks on our training and validation sets by repeatedly: (i) sampling a query uniformly at random from our dataset; (ii) ranking the associated documents using our production ranker; and (iii) generating clicks according to the PBM user model (Eq. 2.1). As in [176], we generate validation clicks proportional to the training / validation split ratio in each dataset (Table 2.1). When sampling clicks according to the PBM, we use the human relevance labels provided by the datasets as ground truth for the document relevance y_d. We use a graded notion of document relevance [5, 7, 26, 90] and add click noise of $\epsilon = 0.1$ to also sample clicks on documents of zero relevance:

$$y_d = \epsilon + (1 - \epsilon) \cdot \frac{2^{s_d} - 1}{2^4 - 1}. \tag{2.10}$$

[1] LightGBM Version 3.3.2, using 100 trees, 31 leafs, and learning rate 0.1.

Table 2.1: Overview of the LTR datasets used in this work.

Dataset	#Features	#Queries	%Train / val / test	#Documents per query				
				min	mean	med.	p90	max
MSLR-WEB30K	136	31,531	60 / 20 / 20	1	120	109	201	1,251
Istella-S	220	33,018	58.3 / 19.9 / 21.8	3	103	120	147	182
Yahoo! Webscope	699	29,921	66.6 / 10 / 23.3	1	24	19	49	139

We follow Joachims et al. [99] and simulate the position bias for a document at rank k after preranking as:

$$o_k = \left(\frac{1}{k}\right)^{\eta}. \tag{2.11}$$

The parameter η controls the strength of position bias; $\eta = 0$ corresponds to no position bias. We use a default of $\eta = 1$. Lastly, we apply an optimization step from [137] and train on the average click-through-rate of each query-document pair instead of the actual sampled raw click data [137, Eq. 39]. This allows us to scale our simulation to millions of queries and multiple repetitions while keeping the computational load almost constant. Our experimental results hold up without this optimization step.

2.5.3 Model implementation and training

We estimate document relevance from features using the same network architecture $g(x_d)$ for both the click model and IPS-based ranker. Similar to [175, 176], we use a three layer feed-forward network with $[512, 256, 128]$ neurons, ELU activations, and dropout 0.1 in the last two layers. We pick the best-performing optimizer[2] and learning rate[3] over five independent runs on the validation set for each model. In all experiments, we train our models on the synthetic click datasets up to 200 epochs and stop early after five epochs of no improvement of the validation loss. We do not clip propensities in the IPS model to avoid introducing bias [1, 99].

2.5.4 Experimental runs

We follow related work and report the final evaluation metrics on the original annotation scores of the test set [1, 99, 137]. We test differences for significance using a two-tailed student's t-test [165], apply the Bonferroni correction [16] to account for multiple comparisons, and use a significance level of $\alpha = 0.0001$. All results reported in this chapter are evaluated over ten independent simulation runs with different random seeds.

We compare five models:

- **IPS / PBM - Naive:** A version of both models that does not compensate for position bias. In this case both models are equivalent (Proposition 2).
- **IPS - True bias:** Pointwise IPS ranker using the true simulated position bias.
- **PBM - Estimated bias:** Neural PBM jointly inferring position bias and document relevance during training.
- **PBM - True bias:** Neural PBM initialized with the true position bias; the bias is fixed during training.
- **Production ranker:** LambdaMART production ranker used to pre-rank queries during simulation.

[2] optimizer $\in \{Adam, Adagrad, SGD\}$

[3] learning rate $\in \{0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001\}$

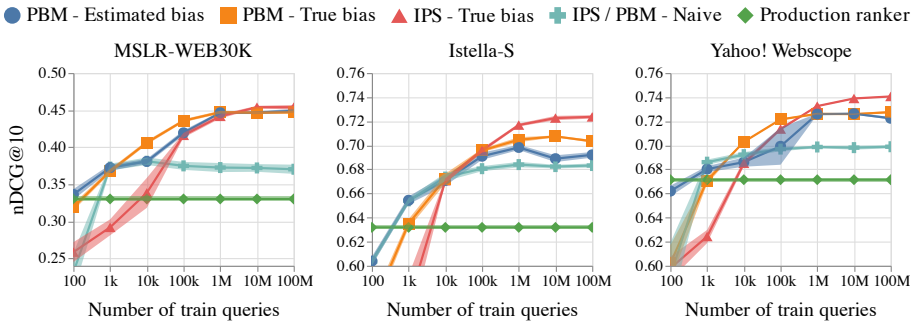


Figure 2.2: Test performance after training on up to 100M simulated queries. All results are averaged over ten runs, and we display a bootstrapped 95% confidence interval.

2.6 Results and Analysis

We examine if the neural click model and pointwise IPS models are empirically equivalent in a semi-synthetic click simulation.

2.6.1 Main findings

Figure 2.2 displays the test performance of all model combinations when training up to 100M simulated queries; full tabular results are available in Table 2.2. Inspecting Figure 2.2, we first note that all approaches improve over the initial rankings provided by the production ranker. The version of both models not correcting for position bias (*IPS / PBM - Naive*) converges to its final, suboptimal, performance after one million clicks. Significantly improving over the naive baseline on two out of three datasets (except *Istella-S*) is the neural click model jointly estimating position bias and relevance (*PBM - Estimated bias*).

Next, we see that providing the *PBM - True Bias* model with access to the correct position bias stabilizes and improves performance significantly over the naive baseline on all datasets. While having a lower variance, the improvements over *PBM - Estimated Bias* are not significant on any of the datasets. The *IPS - True bias* model is less effective than the neural click models for the first 100k clicks but ends up outperforming the click model significantly on two of the three LTR datasets (*Istella-S* and *Yahoo! Webscope*). These differences under idealized conditions between pointwise IPS and the click model are small, but significant. And to our surprise, the neural click model performs worse than the pointwise IPS model, even with access to the true position bias.

In Theorem 1, we prove that click models can recover unbiased document relevance when the position bias is accurately estimated. However, our empirical evaluation indicates a difference between click model and IPS-based approaches, even under the idealized conditions assumed in this setup: *unlike the IPS-based approach, the neural click model may suffer from bias*. Given this observed difference, we conduct further analyses by revisiting the effect of position bias on the magnitude of the click model's loss discussed earlier in Section 2.4.2.

Table 2.2: Ranking performance on the full-information test set after 100M train queries as measured in nDCG and Average Relevant Position (ARP) [99]. Results are averaged over ten independent runs, displaying the standard deviation in parentheses. We mark significantly higher ▲ or lower performance ▼ compared to the **PBM - True bias** model using a significance level of $\alpha = 0.0001$.

Dataset	**Model**	**nDCG@5** ↑		**nDCG@10** ↑		**ARP** ↓	
MSLR-WEB30K	Production	0.301	(0.027) ▼	0.330	(0.024) ▼	49.223	(0.693) ▲
	Naive	0.348	(0.022) ▼	0.370	(0.020) ▼	48.386	(0.538) ▲
	PBM - Est. Bias	0.429	(0.010)	0.449	(0.008)	44.835	(0.274)
	PBM - True Bias	0.428	(0.006)	0.447	(0.006)	44.965	(0.230)
	IPS - True Bias	0.432	(0.011)	0.454	(0.010)	44.418	(0.227)
Istella-S	Production	0.566	(0.012) ▼	0.632	(0.010) ▼	10.659	(0.207) ▲
	Naive	0.616	(0.005) ▼	0.683	(0.005) ▼	9.191	(0.154) ▲
	PBM - Est. Bias	0.629	(0.008)	0.692	(0.007)	10.605	(1.193)
	PBM - True Bias	0.638	(0.003)	0.703	(0.004)	8.911	(0.212)
	IPS - True Bias	0.656	(0.005) ▲	0.724	(0.004) ▲	8.274	(0.141) ▼
Yahoo! Webscope	Production	0.613	(0.012) ▼	0.671	(0.009) ▼	10.439	(0.095) ▲
	Naive	0.647	(0.006) ▼	0.699	(0.004) ▼	10.199	(0.052) ▲
	PBM - Est. Bias	0.673	(0.005)	0.722	(0.003)	9.848	(0.055)
	PBM - True Bias	0.680	(0.004)	0.728	(0.003)	9.812	(0.035)
	IPS - True Bias	0.695	(0.001) ▲	0.741	(0.001) ▲	9.658	(0.011) ▼

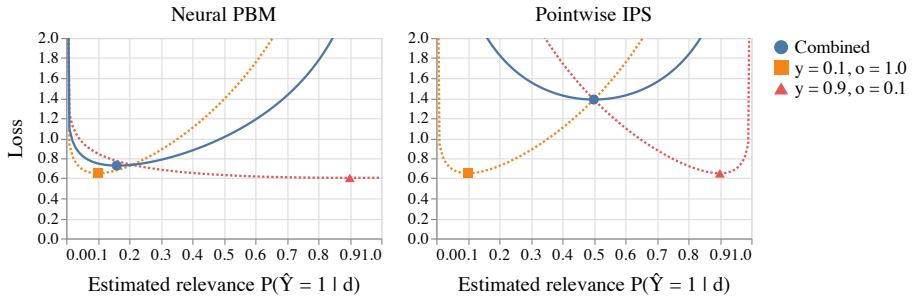


Figure 2.3: Visualizing the loss and estimated document relevance of two documents when calculated separately (dotted lines) and combined (solid line).

2.6.2 Further analyses

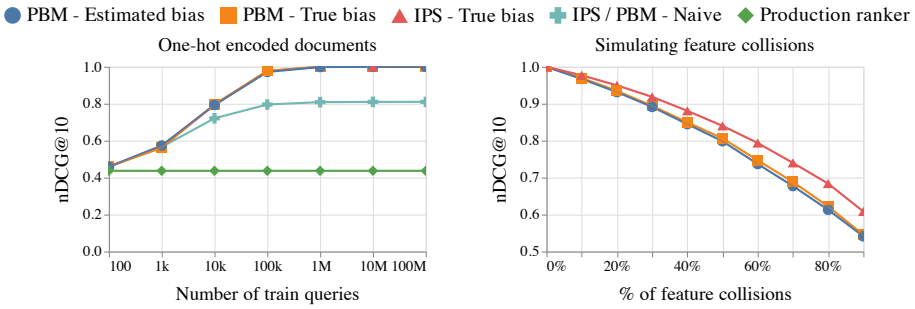
Our first hypothesis to explain the lower performance of the neural click model concerns hyperparameter tuning. Section 2.4.2 shows that the click model loss decreases with an increase in position bias. Through manual verification, we find that items at lower positions have smaller gradient updates, affecting the choice of learning rate and the number of training epochs. While this is a concern when using SGD, our extensive hyperparameter tuning and use of adaptive learning rate optimizers should mitigate this issue (Section 2.5.3). Given that the observed performance difference persists despite extensive optimization, we reject this hypothesis.

Instead, we hypothesize that higher ranked items might overtake the gradient of lower ranked items, given their higher potential for loss reduction. This case might occur when encountering two documents with similar features but different relevance. The item at the higher position could bias the expected relevance towards its direction. This is indeed what we find when simulating a toy scenario with two documents in Figure 2.3. There, we display one relevant but rarely observed document (red triangle) and one irrelevant but always observed item (orange square). Both click model and IPS approaches converge to the correct document relevance when computing the loss for each item separately, but when computing the combined average loss, the IPS approach converges to the mean relevance of both items while the click model is biased towards the item with the higher examination probability.

One can frame this finding as an instance of *model misfit*. Theorem 1 demands a separate parameter $\hat{y}_d$ for each query-document pair, but by generalizing over features using the relevance network g, we might project multiple documents onto the same parameter $\hat{y}_d$, which might be problematic when features do not perfectly capture item relevance. We test our hypothesis that the click model's gradient updates are biased towards items with higher examination probabilities with three additional experiments.

2.6.3 No shared document features

First, we should see an equivalent performance of both approaches in a setting in which documents share no features since the gradient magnitude should not matter in this



(a) Test performance when documents share no features.

(b) Gradually introducing feature collisions between documents.

Figure 2.4: Experiments on one-hot encoded documents. All results are averaged over ten independent runs. We display a bootstrapped 95% confidence interval.

setting. We create a fully synthetic dataset of 10,000 one-hot encoded vectors with uniform relevance scores between 0 and 4. To avoid feature interactions, we reduce the depth of the relevance network g to a single linear layer. We find in Figure 2.4a that indeed both approaches are able to recover the true document relevance. Every document in the validation or test set appears once in the train dataset, thus achieving a perfect ranking score (e.g., $nDCG@10 = 1.0$) is possible in this setting.

2.6.4 Feature collisions

Second, gradually forcing documents to share features by introducing random feature collisions into our one-hot encoded dataset should lead to a stronger drop in performance for the click model. At the start of each simulation, we use a modulo operation to assign a share of documents based on their id on to the same one-hot encoded feature vectors. Figure 2.4b shows that both approaches perform equivalently when each document has its own feature vector. But when gradually introducing collisions, *PBM - Estimated bias* and *PBM - True bias* deteriorate faster in performance than *IPS - True bias*.

2.6.5 Mitigating position bias

A last interesting consequence is that the problem of neural click models biasing relevance estimates toward items with higher examination probabilities should get worse with an increase in (known) position bias. Simulating an increasing position bias and supplying the examination probabilities to both approaches on *Istella-S* shows that IPS can recover consistently from high position bias, while the click model deteriorates in performance with an increase in position bias (Figure 2.5).

In summary, we have found strong evidence that when encountering documents of different relevance but similar features, the neural click model biases its relevance estimate towards items with higher exposure.

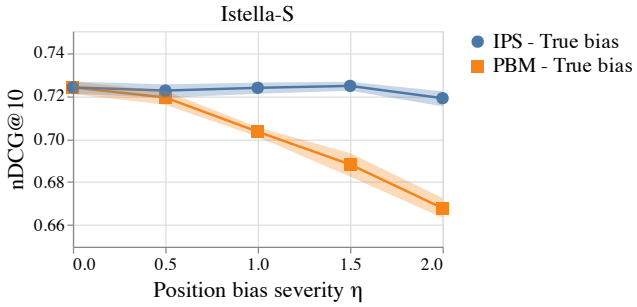


Figure 2.5: Simulating an increasing (known) position bias. We report test performance after 100M clicks over 10 independent runs.

2.7 Conclusion

In this chapter, we have considered **RQ1** on whether recent neural click models and pointwise IPS rankers are equivalent for pointwise learning-to-rank from position-biased user clicks. We show theoretically and empirically that neural click models and pointwise IPS rankers achieve equal ranking performance when the true position bias is known, and relevance is estimated for each item separately. However, we also find small but significant empirical differences in ranking performance, indicating that the neural click model may be affected by position bias when learning from shared and potentially conflicting document features. Therefore, we answer our first research question negatively, as our empirical evaluation indicates that both approaches handle conflicting document features differently.

Given the similarity of the neural PBM used in this chapter to current industry trends [72, 85, 195, 202], practitioners should investigate whether their model architecture is vulnerable to the described bias, especially when representing items using a small set of features or low-dimensional latent embeddings. Potential diagnostic tools include simulating synthetic clicks or training a related pointwise IPS method to test for performance improvements.

We emphasize that the findings in this chapter are specific to our neural PBM and simulation setup, and we make no claims about other architectures, such as additive two-tower models [195]. Critically, this chapter focuses solely on comparing relevance prediction between the two methods. However, for use cases in which the ability to predict clicks is equally important, the neural PBM's bias towards more examined documents might have different implications that warrant further investigation. In addition, future work may investigate connections and differences between IPS and click models beyond the pointwise ranking setting and under more sophisticated conditions, such as mixtures of user behavior and bias misspecification.

These limitations motivate our next chapter, where we evaluate a range of unbiased learning-to-rank methods, including an additive two-tower model [195] and an EM-based click model [184], as well as pointwise and listwise IPS approaches, outside of simulation, on the largest real-world click dataset from the Baidu search engine.

2.8 Reproducibility

We share the code for this chapter at: https://github.com/philipphager/ultr-cm-vs-ips/

3

Unbiased Learning to Rank Meets Reality: Lessons from Baidu's Large-Scale Search Dataset

Large-scale datasets of real user clicks for unbiased learning-to-rank are scarce [209]. Therefore, especially in academia, researchers often resort to semi-synthetic simulations [5, 48, 75, 99, 137, 140, 176, 208]. The most common simulation setup uses traditional learning-to-rank datasets comprised of pre-computed query-document feature vectors and relevance annotations to simulate synthetic user clicks [99]. While simulations are invaluable for verifying theory, they have inherent limitations: (i) Simulated user behavior is typically based on simple, known models, while real-world users are more complex and noisy, limiting our ability to observe novel phenomena and challenges [48]. (ii) The LTR datasets commonly used for simulation are small and dated, potentially tying the field to ranking with numerical input features (e.g., BM25, TF-IDF, and PageRank scores [24, 41, 145]), while much of modern IR research has shifted toward automatic feature extraction from raw text using transformers [61, 104, 118, 133, 207]. (iii) Simulations typically focus solely on clicks, whereas modern search engine providers track a variety of signals, including dwell time, scrolling, and skipping behavior [209]. These limitations raise the fundamental question of whether ULTR methods work well on contemporary real-world datasets.

The Baidu-ULTR dataset [209], comprising over 1.2 billion search sessions from the Chinese search engine Baidu, offers a rare opportunity to address this question. Initial experiments by Zou et al. [209] showed that prominent ULTR methods performed no better than a ranking baseline trained without bias correction. Given the substantial implications for the field, we conduct a comprehensive reproducibility study, asking:

RQ2 Do unbiased learning-to-rank methods that work well in simulation actually improve ranking performance on the largest real-world search dataset?

In this chapter, we reproduce six well-established ULTR methods for position bias correction on the Baidu-ULTR dataset and compare them across different input representations, ranking loss functions, and model combinations.

This chapter was published as P. Hager, R. Deffayet, J.-M. Renders, O. Zoeter, and M. de Rijke. Unbiased

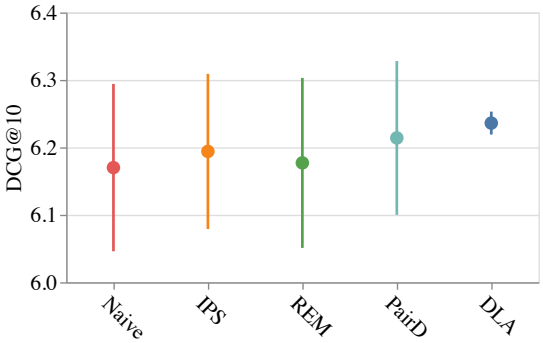


Figure 3.1: The original experiments on Baidu-ULTR show that none of the four compared ULTR methods outperform a naive method not correcting for position bias. Data from [209, Table 6] and visualization by us.

3.1 Introduction

Historically, the field of learning-to-rank employed human experts to annotate the relevance of search results to create training data for ranking models [41, 42, 145]. As expert labeling is expensive [24], infeasible in certain applications [183], and potentially misaligned with user preferences [159], many practitioners seek to leverage implicit user feedback, often collected in the form of clicks. However, clicks are usually a biased signal of relevance, as they often depend on the position of an item [97], its surrounding items [39, 206], or even the user's trust in the system to place relevant items on top [2, 97, 176]. Over the years, the field of unbiased learning-to-rank (ULTR) has proposed methods to mitigate biases when training ranking models on click data [76].

ULTR methods, especially in academic research, are often evaluated in synthetic or semi-synthetic setups [5, 99, 138, 175, 176]. The most common semi-synthetic setup goes back to the seminal work by Joachims et al. [99]. The setup uses learning-to-rank (LTR) datasets from web search engines containing features and expert judgments of each query-document pair [24, 41, 145] and simulates clicks and biases based on synthetic user models (e.g., the position-based model (PBM) [39, 153] or the cascade model [39]).

While the semi-synthetic simulation setup has demonstrated the effectiveness of many unbiased learning-to-rank (ULTR) methods, it is often unclear to academic researchers how these methods fare in a real-world setup. Recently, Zou et al. [209] released Baidu-ULTR, an extensive real-world dataset for web search. It comprises over 1.2 billion user sessions with click data and $397,572$ annotated query-document pairs for evaluation. The dataset contains rich user feedback, including clicks, dwell time, whether a document was scrolled off-screen, and whether the user returned to the result page after clicking an item. In contrast to the classic LTR datasets used in semi-synthetic simulation setups, the dataset does not contain pre-computed ranking features, but the

learning to rank meets reality: Lessons from Baidu's large-scale search dataset. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024.

original search query, title, and abstract of each result (for privacy reasons in tokenized form, with a private vocabulary). Therefore, the dataset enables training and evaluating transformer-based rankers, such as MonoBERT [133] or MonoT5 [143].

Zou et al. [209] train a MonoBERT ranker and provide baseline results of this model fine-tuned on four standard ULTR methods: inverse propensity scoring (IPS) [99, 183], regression expectation-maximization (RegressionEM) [184], the dual learning algorithm (DLA) [5], and pairwise debiasing (PairD) [90]. Interestingly, Zou et al. [209] find that these prominent ULTR methods overall do not perform significantly better than a naive BERT model trained on clicks without any bias correction,[1] as we report in Figure 3.1. In subsequent work, the Baidu-ULTR dataset was used at multiple ranking competitions (the WSDM Cup 2023 [210, 211] and NTCIR's ULTRE-2 [132]), in which multiple teams reported noticeably higher ranking performance, with and without ULTR techniques. *It therefore remains unclear whether the observed improvements were due to unbiased learning-to-rank, and ultimately whether ULTR techniques improve performance on this dataset.*

In this work, we reproduce and extend the experiments by Zou et al. [209], in light of several considerations:

(1) The initial findings of ULTR methods bringing barely any benefits on the largest real-world dataset available for unbiased learning-to-rank (ULTR) have substantial implications for the field and warrant more scrutiny.

(2) Three competitions have been conducted on the Baidu-ULTR dataset, with participants reporting vastly improved ranking results, though not necessarily using ULTR. A preliminary experiment by us revealed that a random ranking of the annotated test dataset achieves a DCG@10 ≈ 6.69, which is higher than all baselines in Figure 3.1 and suggests potential issues with the original experiments.

(3) According to the official codebase that comes with the Baidu-ULTR dataset,[2] the original comparison focuses on pointwise versions of the compared ranking methods (except for pairwise debiasing). Therefore, it remains open how pointwise, pairwise, and listwise ULTR methods compare on this dataset.

(4) And lastly, a closer look at the Baidu-ULTR dataset and the original codebase revealed some disputable design decisions. IPS, for example, requires an estimate of position bias on the given dataset, while the creators of the Baidu-ULTR dataset confirmed using a hardcoded position bias from the ULTRA library in their experiments [173]. We also found that placeholder documents make up almost 20% of the Baidu-ULTR dataset (see Section 3.3), with an unknown impact on the trained models.

By revisiting the experiments reported by Zou et al. [209] and the participants of the WSDM cup, we address the following research questions:

[1] Zou et al. [209] find that DLA outperforms a naive baseline on frequent head queries.
[2] https://github.com/ChuXiaokai/baidu_ultr_dataset/

RQ2.1 Does unbiased learning-to-rank improve performance on the Baidu-ULTR dataset over naive, non-debiasing models?

RQ2.2 How do ULTR methods fare against each other, and how do ranking losses and input features affect their performance?

RQ2.3 Can ULTR methods be applied during language model training, and do they bring improved performance?

We answer **RQ2.1** negatively as we cannot find robust improvements in ranking performance in a fair comparison between ULTR-based and naive methods. For **RQ2.2**, we report minor but existing differences between ULTR methods and showcase large and reliable differences due to ranking losses and input features. Regarding **RQ2.3**, we find that while certain ULTR methods improve over their naive counterpart if applied during language model training, the interactions of ULTR and transformer-based models are not well understood and can even lead to decreased performance.

Besides answering our research questions, our contributions are:

- We release two smaller, cleaned, and pre-processed datasets derived from Baidu-ULTR,[3] along with transformer-based query-document embeddings to enhance the reproducibility of our work and ease access to Baidu-ULTR.
- We publish highly optimized Jax [18, 94] implementations of various standard ULTR methods to enable ULTR researchers to leverage the vast scale of the Baidu-ULTR dataset effectively.[4]
- We tune six ULTR methods and three naive ranking methods on transformer-based embeddings and learning-to-rank features.
- We train six MonoBERT models from scratch, including listwise and ULTR-based loss functions; our model weights are public.[5]

3.2 Related Work

3.2.1 Unbiased learning to rank

The field of unbiased learning-to-rank can be broadly divided into click modeling and counterfactual learning-to-rank. Click modeling encodes assumptions on user behavior into probabilistic models [25, 39, 56]. Assumed effects, such as position bias or item relevance, are represented as (latent) variables that are jointly inferred via maximum likelihood estimation [37]. Notable examples include the *position-based model* (PBM) [39, 153], which assumes that users click only on positions they examine and items they find relevant, and the *cascade model* [39], which assumes that users scroll from top to bottom, click on the first relevant item, then leave the page. In

[3] https://huggingface.co/datasets/philipphager/baidu-ultr_uva-mlm-ctr
[4] https://github.com/philipphager/ultr-reproducibility
[5] https://github.com/philipphager/baidu-bert-model

recent years, more complex and flexible click models have been proposed using neural architectures, semantic embeddings, and bias features beyond position [17, 28, 195, 206]. We implement two neural versions of the original PBM: RegressionEM [184] and an additive two-tower model popular in industry applications [72, 200, 202].

The counterfactual learning-to-rank community has historically employed simpler user models such as the PBM [99, 183] or the affine model for trust bias [176] and focused on training more effective ranking models, mitigating biases with inverse propensity scoring [99, 138, 183]. Our work includes an extended binary cross-entropy loss [13, 157] as a pointwise IPS baseline and an extended softmax loss as a listwise IPS baseline [5, 19]. Both methods require position bias estimations, for which we release the implementation of three intervention harvesting methods (more in Section 3.3.2). We also include two counterfactual methods jointly estimating position bias and item relevance, the dual learning algorithm (DLA) [5] and pairwise debiasing (PairD) [90]. We introduce all unbiased learning-to-rank methods used in this chapter in Section 3.4.

3.2.2 ULTR on the Baidu-ULTR dataset

Most prior work on the Baidu-ULTR dataset has revolved around three public competitions: Two tracks at the WDSM Cup 2023 – *unbiased learning-to-rank* [211] and *pretraining for web search* [210] – and the ULTRE-2 track at NTCIR [132].

In the ULTR track of the WSDM cup, participants were restricted to train ranking models using click data. Two of the top three teams ended up applying ULTR techniques [33, 197], notably the winning team which employed a two-tower model with a softmax loss [33]. However, participants did not compare with non-ULTR methods (as the main objective was to win a competition). All three top-performing teams incorporated a BERT-based cross-encoder [133], with the first two teams training models from scratch [29, 33]. Chen et al. [29] voice concerns about the performance of their BERT model trained using the officially released starter-kit, as it plateaued around a DCG@10 ≈ 7. Their remark has led us to train our own BERT cross-encoder models from scratch.

The second task of the WSDM Cup allowed participants to pre-train language models from clicks before fine-tuning on expert annotations. All top teams combined the output of their BERT models with traditional LTR features and tuned gradient-boosting models on annotations [113, 116, 166]. Only the third-placed team incorporated a conventional ULTR approach with a softmax ranking loss and IPS during BERT pre-training.

In the NTCIR 17 ULTRE-2 track [132], the organizers released a subset of the data ($\approx$ 1 million sessions) to make Baidu-ULTR more accessible, using the best-performing BERT model trained by Li et al. [116] during the WSDM Cup to create query-document embeddings in combination with traditional lexical matching features.[6] In their baseline experiments, they find that DLA models perform better than non-ULTR models trained with a pointwise loss but only marginally better than those trained with a listwise loss. The only participating team, Yu et al. [196], proposed to alleviate selection bias on items

[6]Note that the first ULTRE-1 task at NTCIR 16 was not yet conducted on Baidu-ULTR but on a semi-synthetic simulation setup [201].

that are relevant but never clicked by using a DLA model to re-annotate non-clicked documents. However, they do not compare with other ULTR and non-ULTR baselines.

Overall, participants in the three cups reported noticeably higher ranking performance than the original experiments by the Baidu-ULTR dataset authors [209], and identified helpful techniques, such as re-weighting queries to address the long-tail query distribution [166], negative sampling of documents [166], ignoring clicks on items displayed for a short time [116], or pseudo-relevance feedback [196]. However, it remains unclear whether observed improvements stemmed from debiasing the click feedback or such orthogonal techniques. In order to identify the contribution of click debiasing to the results, we extend the work by Niu et al. [132] and Zou et al. [209] to larger datasets, more ULTR methods, fairer baselines, and investigate the interactions of ULTR with language model training. Next, we introduce the dataset used in this chapter in more detail.

3.3 Overview of Baidu-ULTR

3.3.1 Description of the data

The Baidu-ULTR dataset [209] contains more than 1.2 billion search sessions (split into $2,000$ dataset partitions) with user clicks and $7,008$ annotated queries for evaluation. The dataset was collected in April 2022 by randomly sampling the search traffic of Baidu [209], reflecting the long-tail query distribution of real user traffic. The dataset contains logged queries, document titles, and abstracts of the search results presented to the user. The authors tokenized all released text with a private vocabulary for user privacy. In addition to tokenized text, the dataset also contains a multitude of logged user interactions, including clicks, dwell-time, and skipping behavior, as well as item presentation features, including document height, item type, and ranking position. In this work, we solely focus on item position and user clicks.

To ease access to the vast Baidu-ULTR dataset for academic research, we release two smaller datasets: A *language modeling dataset*, comprised of the first 125 partitions of the dataset for training transformer models from scratch, and a smaller *reranking dataset* comprised of four partitions to compare ULTR on pre-computed query-document features. In the following, we describe the preprocessing common to both datasets and analyze the properties of our reranking dataset.

Pre-processing

(i) We use the md5 hash of the query tokens and document URL, respectively, as query and document identifiers. (ii) We discovered that over 20% of the Baidu-ULTR dataset comprised only two title-abstract token combinations. To avoid these documents polluting model training, we remove the *what other people searched* item from the ranking ($\approx 9\%$ of documents) and skip all documents with only a dash in the title, indicating no available content ($\approx 13\%$ of documents).[7] Note that documents displayed

[7]While Baidu-ULTR was tokenized with a private vocabulary, Zou et al. [209] confirmed to us that the tokens: $[3742, 0111492, 0112169, 015061, 0116905]$ translate to *what other people searched* and token 21429 translates to "-" indicating missing content.

after a skipped document keep their original position and do not, e.g., move to the position of a skipped document. (iii) Third, we drop all queries with less than five documents to display, affecting $\approx 2\%$ of the train queries and $\approx 0.3\%$ of annotated queries, leaving a remainder of 6,985 annotated test queries. In contrast to previous work [132], we do not remove queries without any clicks. While this is common in ULTR as sessions without clicks do not contribute to popular pairwise ranking losses [99], they are essential to train and evaluate methods predicting calibrated click probabilities.

Analysis of the reranking dataset

The reranking dataset comprises three dataset partitions for training (≈ 1.8 million user sessions, ≈ 11.7 million query-document pairs) and one partition for validation and testing ($\approx 590k$ user sessions, ≈ 4.8 million query-document pairs). Table 3.1 gives an overview of the statistics of the reranking dataset. While the dataset is a fraction of Baidu-ULTR, we highlight that it is still substantially larger than existing ULTR datasets [5] and contains more unique queries and query-document pairs than the LTR datasets commonly used in semi-synthetic simulation for ULTR [24, 41, 42, 145].

Besides query-document text and identifiers (as described in the preprocessing section), the reranking dataset also contains three sets of pre-computed query-document features: the CLS token of the BERT cross-encoder released by Zou et al. [209], the CLS token of a BERT cross-encoder trained by us in the same fashion as the original model, and classic lexical learning-to-rank (LTR) features, including BM25 [155], Tf-IDF [154], and query-likelihood (with Jelinek-Mercer and Dirichlet smoothing [31]). We compute the required inverted index for the methods above on the *language modeling dataset* that we use to train our BERT cross-encoder for fair comparison. As we restrict ourselves to solely training on click data and only use the expert annotations in the final evaluation, we compute all lexical matching features with default parameters instead of tuning them on annotations. Also, LTR parameters published by Chen et al. [29] do not lead to major improvements over untuned default parameters in our setting. We publish two versions of the dataset on Hugging Face, one using the Baidu cross-encoder[8] and one with our cross-encoder[9] with a list of all pre-computed lexical ranking features and their respective hyperparameters.

As the reranking dataset is a random sample of the larger (preprocessed) dataset, we can use it to analyze the basic properties of Baidu-ULTR. After preprocessing, the average search session contains 8 documents, with an average of 0.68 clicks per session (number of clicks/number of sessions) and $\approx 46.5\%$ of sessions containing at least one click (see Table 3.1). Notably, we measure the overlap between queries (md5 hash of query tokens) in the click dataset and the annotated test set and find that only 12.7% of annotated queries occur in the training dataset. While they appear during training, they make up $< 0.1\%$ of the training dataset.

[8] https://huggingface.co/datasets/philipphager/baidu-ultr_baidu-mlm-ctr
[9] https://huggingface.co/datasets/philipphager/baidu-ultr_uva-mlm-ctr

Table 3.1: Description of the reranking dataset we derive from Baidu-ULTR [209]; statistics after preprocessing.

		Implicit feedback	
	Annotations	train	test
Unique queries	6,985	1,378,901	501,215
Unique documents	381,552	9,455,953	3,557,825
Total query/doc pairs	382,038	11,715,447	4,209,900
Total sessions	-	1,779,017	593,930
Total impressions	-	14,526,276	4,848,878
Avg. docs per session		8.165	
Avg. clicks per session		0.688	
Avg. clicks per document		0.084	
% of sessions with $\geqslant 1$ click		46.581%	
% of sessions with $\geqslant 2$ clicks		13.082%	
% of train queries occurring in the annotated set		0.064%	
% of annotated queries occurring in the train set		12.713%	

3.3.2 Position bias

Standard counterfactual learning-to-rank methods using IPS require an estimation of position bias, which we estimate on our reranking dataset. We implement three intervention harvesting techniques that leverage the co-occurrence of the same query-document pair at different ranks [3]. Intervention harvesting mines query-document pairs logged in various positions, ideally due to distinct rankers in an A/B test ranking items differently. We can use this natural variability of encountering a document in different positions to estimate bias using click ratios between neighboring positions (Adjacent Pair), each position and a fixed rank (Pivot Rank), or between all ranking positions (All Pairs). We refer the reader to Agarwal et al. [3] for a detailed introduction to intervention harvesting.

We also estimate position bias using RegressionEM (REM) [184], which leverages query-document embeddings. REM estimates position bias over co-occurrences of query-document pairs with similar features rather than strictly identical query-document pairs, like intervention harvesting. In our case, we use the semantic embeddings of our naive BERT cross-encoder. Figure 3.2 displays our propensity estimations and the click-through rate per rank. The estimated position bias is broadly consistent across methods. REM and intervention harvesting arrive at similar estimations from different query-document representations. Our finding suggests the presence of a noticeable, top-heavy position bias affecting user behavior on Baidu-ULTR for which ULTR methods should be helpful.

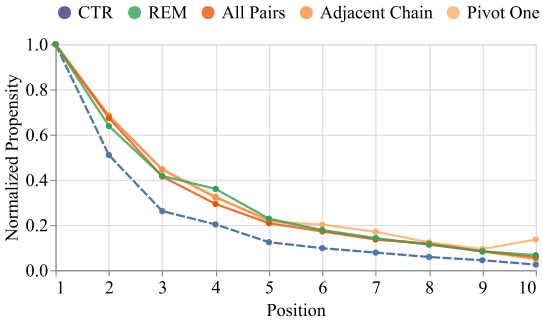


Figure 3.2: Position bias as estimated by RegressionEM and three intervention harvesting methods compared to the mean CTR. Propensities were normalized by position one.

3.4 Unbiased Learning-to-Rank Methods

We now introduce the different ULTR methods used throughout this work. Similarly to Zou et al. [209], we restrict ourselves to methods that address position bias according to the PBM and focus on benchmarking the learning algorithm used by each method.

First, we introduce the notation we will use to describe each method. Let q be a user query and D_q the ordered list of documents for q. Let $d \in D_q$ be a document in the list displayed at position k. We use $R_{q,d}$, E_k and $C_{q,d,k}$, respectively, to denote the random events for: (i) document d being relevant for query q, (ii) position k of the result page being examined by the user, and (iii) document d placed in position k for query q being clicked. We assume users follow the position-based model (PBM), meaning that users click only if they observed the position k of an item and deemed the displayed document d to be relevant:

$$P(C_{q,d,k}) = P(E_k) \times P(R_{q,d}). \tag{3.1}$$

In the following, we use $\mathcal{L}$ to reference a ranking loss function; in this work, either binary cross-entropy, softmax cross-entropy [19], or LambdaRank [21]. All ranking methods aim to estimate document relevance $r(q, d) = P(R_{q,d})$ and optionally examination $e(k) = P(E_k)$ (or related quantities) from observed clicks $c \in \{0, 1\}$. We use $\tilde{x}$ to denote estimators of a quantity x. We now define the methods compared in our study:

Naive: is the common ULTR term for applying a ranking loss without any position bias correction. This means that we naively interpret clicks as positive/negative feedback:

$$\mathcal{L}^{\text{Naive}} = \mathcal{L}(\tilde{c}(q, d); c). \tag{3.2}$$

We implement a pointwise version of this model using binary cross-entropy and two listwise versions using softmax cross-entropy [19] and LambdaRank [21], respectively.

Two-Tower [195, Two-Tower Model]: jointly learns the maximum-likelihood parame-

ters of a relevance model $\tilde{r}$ and an examination model $\tilde{e}$, mirroring the PBM:

$$\mathcal{L}^{\text{TwoTower}} = \mathcal{L}\left(\sigma\left(\tilde{e}(k) + \tilde{r}(q,d)\right); c\right). \tag{3.3}$$

We use the additive formulation [195, 200] with $\tilde{r}$ and $\tilde{e}$ representing logits, σ the sigmoid function, and $\mathcal{L}$ the binary cross-entropy loss.

RegressionEM [184, Regression expectation-maximization]: learns a maximum likelihood estimate of relevance and examination parameters through the EM algorithm [50]:

$$\mathcal{L}^{\text{REM}} = \mathcal{L}(\tilde{r}(q,d); c + (1-c)\overline{r}(q,d)) + \mathcal{L}\left(\tilde{e}(k); c + (1-c)\overline{e}(k)\right), \tag{3.4}$$

where $\overline{r}(q,d) = \frac{\tilde{r}(q,d)(1-\tilde{e}(k))}{1-\tilde{r}(q,d)\tilde{e}(k)}$ and $\overline{e}(k) = \frac{\tilde{e}(k)(1-\tilde{r}(q,d))}{1-\tilde{r}(q,d)\tilde{e}(k)}$ are the posterior relevance and examination probabilities. In practice, we use binary cross-entropy as the base loss and replace the original EM procedure with a numerically stable gradient-based version suggested in TF-Rank [141] by performing the posterior computation with logits.

IPS [99, Inverse propensity scoring]: Given known examination probabilities e_k, IPS re-weights the click labels by the propensity (i.e., normalized examination probability) of the current position:

$$\mathcal{L}^{\text{IPS}} = \mathcal{L}\left(\tilde{r}(q,d); \frac{\max(\tau, e_1)}{\max(\tau, e_k)} c\right). \tag{3.5}$$

In our experiments, we use propensities estimated by the AllPairs intervention harvesting method [3] and reduce variance by clipping propensities to a minimum value of $\tau = 0.1$ [99]. We implement a pointwise [13, 157] and listwise IPS variant [5, 19].

DLA [5, Dual learning algorithm]: learns tunable relevance scores with fixed propensities and tunable propensities with fixed relevance scores, i.e., applying IPS twice:

$$\mathcal{L}^{\text{DLA}} = \mathcal{L}\left(\tilde{r}(q,d); \frac{\tilde{e}(1)}{\tilde{e}(k)} c\right) + \mathcal{L}\left(\tilde{e}(k); \frac{\tilde{r}(q,d_1)}{\tilde{r}(q,d)} c\right), \tag{3.6}$$

where d_1 is the document ranked first on the current result page. We use softmax cross-entropy as the base loss and perform a softmax-normalization of examination and relevance probabilities $\tilde{r}$ and $\tilde{e}$ within a result page, as in the original paper [5].

PairD [90, Pairwise debiasing]: learns positive $\tilde{e}^+$ and negative $\tilde{e}^-$ propensities (i.e., corresponding to clicks and non-clicks, respectively) with a constraint on the norm of the learned propensities:

$$\mathcal{L}^{\text{PairD}} = \frac{\mathcal{L}(\tilde{r}(q,d); c)}{\tilde{e}^+(k)\tilde{e}^-(k)} + |\tilde{e}^+|_1 + |\tilde{e}^-|_1. \tag{3.7}$$

In practice, we use the L_1-norm for propensity regularization and learn the relevance scores using the LambdaRank loss [21], as in the original paper [90]. Note that the theoretical validity of this method under non-trivial position bias has been challenged by Oosterhuis [135]. Still, we include it in our comparison as it has demonstrated strong empirical performance in past comparisons [7].

3.5 Experimental Setup

3.5.1 Training and evaluation procedure

All models are trained on our reranking dataset, and we use a 50/50 random split of our test click dataset (see Table 3.1) for validation and testing. We evaluate ranking performance on $6,985$ annotated queries, where experts rated each query-document pair's relevance on a scale from 0 to 4. We refer to Zou et al. [209] for more details on the annotation process. In this chapter, we do not use available side information such as dwell-time, off-screen scrolling, and returns to the result page. Instead, we focus on query and document content, document position and click label as our training data.

We measure ranking performance on the annotated set using discounted cumulative gain (DCG) at different truncation levels and mean reciprocal rank (MRR) at 10, as well as negative log-likelihood (NLL) for click prediction.

3.5.2 Models

Traditionally, unbiased learning-to-rank practitioners train small ranking models with LTR features such as BM25 [155] or TF-IDF [154] as input [5, 138, 176]. To investigate the role of semantic embeddings and the interaction of ULTR with language model training on such a large dataset, we train two types of models: transformer-based *language models* trained from scratch following the MonoBERT cross-encoder [133], and multi-layer perceptron-based *reranking models*, that take as input either traditional LTR features or fixed query-document embeddings obtained by the language models.

Language models

We train cross-encoders on the Baidu-ULTR dataset from scratch in Jax [18], building on the FlaxBERT implementation from Hugging Face.[10] The input to our BERT models is: `[CLS] query [SEP] doc_title [SEP] doc_abstract`. Following [33, 209], we truncate the input to a maximum length of 128 tokens. Compared to the original BERT training scheme [51], we keep the masked-language modeling task (masking 30% of input tokens at random) but discard the next-sentence prediction task. Instead, we follow the MonoBERT [133] setup and apply a linear layer to the BERT CLS token to output a click prediction score. We train multiple models with different click-based loss functions: naive click prediction with a pointwise (binary cross-entropy) and a listwise (softmax cross-entropy) loss, pointwise and listwise IPS, a pointwise two-tower loss, and a listwise DLA loss (see Section 3.4). All loss functions were built on top of the Rax library [94].

We keep the architecture of the original $\text{BERT}_{\text{base}}$ model, using 12 transformer layers, 12 attention heads, 768 output dimensions, and, following Chen et al. [33], a vocabulary size of $22,000$. All language models are trained with a batch size of 256 (4×64) for 2 million gradient steps, i.e., on 512 million documents. Training each base model on four GPUs (NVIDIA H100-80GB) takes around 46 hours, which is a speed-up of almost 50% compared to an early PyTorch implementation of ours using

[10] https://huggingface.co/docs/transformers/model_doc/bert#transformers.FlaxBertForPreTraining

the exact same library, architecture, and hardware. We use the AdamW optimizer [121] with a fixed learning rate of 5×10^{-5} and a weight decay of 0.01. Given the substantial compute invested to train each language model, we rely on prevalent default parameters for BERT (listed in our repository).

Reranking models

To investigate the interactions of ULTR with semantic query-document embeddings and traditional LTR features on Baidu-ULTR, we train several smaller feed-forward networks as reranking models, as is common in the ULTR community [5, 138, 175]. Our reranking models are composed of linear layers with ReLU activations and, optionally, dropout regularization. We found no benefit in applying Layer or Batch Normalization, as our BERT embeddings are already normalized. As LTR features can span a large range, we find that scaling features with $\log1\text{p}(x) = \log_e(1 + |x|) \odot \text{sign}(x)$ before the first layer, as suggested by Qin et al. [146], works well. Note that the feed-forward network takes the query-document features as input and, depending on the ULTR method, outputs a relevance or click prediction. We add a single learnable model parameter per position for methods that jointly estimate position bias. In contrast to models leveraging multiple bias features [195, 201], we found no additional benefit by using a multi-layer perceptron for position bias estimation in our setting.

3.5.3 Hyperparameter tuning

We perform extensive hyperparameter tuning for a fair comparison of the reranking models in our experiments. Given the immense combinatorial space of hyperparameters, methods, and datasets, we adopt an incremental hyperparameter tuning strategy as advocated by Godbole et al. [64]. First, we tune our model architecture per set of input features based on the pointwise naive model. We tune the number of hidden dimensions $\in \{64, 128, 256, 512, 1024\}$ and the number of layers $\in \{2, 3, 4, 5\}$ over three random seeds. As both network depth and width can interact with the learning rate, we tune each parameter combination over three learning rates $\in \{0.001, 0.0005, 0.0001\}$, i.e., we treat the learning rate as a nuisance parameter [64]. We adopt this incremental tuning procedure as we found no major discrepancies in model architecture between ULTR methods but instead between sets of input features. Subsequently, we adopt the architecture of the pointwise naive model per dataset for all other methods.

A five-layer perceptron with 512 hidden dimensions yields optimal results for our LTR features, while a five-layer perceptron with 256 dimensions performs best for the Baidu BERT embeddings. Additionally, a two-layer perceptron with 256 dimensions was the most suitable choice for our BERT embeddings. Given each base model architecture, we tune the final dropout $\in \{0, 0.3\}$ and learning rate $\in \{0.001, 0.0005, 0.0003, 0.0001\}$ for each method and dataset over three random seeds. We use the AdamW optimizer [121] with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 1e^{-8}$. Most models work best with a learning rate of 0.0001 and do not benefit from dropout. The full list of hyperparameters is available in our online repository. Given the final hyperparameters for each method, we train all methods for 50 epochs, stopping early after five epochs of no improvement of the validation loss computed on clicks. Lastly, while this

tuning step improved the performance of all methods, our main findings are consistent across many different hyperparameter combinations.

3.6 Results

Before presenting our findings, we recall our research questions:

RQ2.1 Does unbiased learning-to-rank improve performance on the Baidu-ULTR dataset over naive, non-debiasing models?

RQ2.2 How do ULTR methods fare against each other, and how do ranking losses and input features affect their performance?

RQ2.3 Can ULTR methods be applied during language model training, and do they bring improved performance?

3.6.1 Unbiased learning-to-rank yields no or tiny improvements on Baidu-ULTR

In our first experiment, we train reranking models with pointwise and listwise ULTR loss functions and their respective naive, non-debiasing counterparts on three different types of input features: the CLS token of Zou et al. [209]'s pointwise cross-encoder, the CLS token of our pointwise cross-encoder, and learning-to-rank features (see Section 3.3.1). We list comprehensive results in Tables 3.3, 3.4, and 3.5. First, we focus on the ranking performance on expert annotations as measured in DCG and MRR, Figure 3.3 displays the DCG@10.

At first glance, Figure 3.3 shows that we can get a much higher ranking performance from our cross-encoder than the initially released BERT cross-encoder, echoing previous work [132, 201]. Models trained on LTR features and even of a trivial baseline assigning random scores (grey dotted line) to the annotated documents beat all models trained on the original Baidu BERT features.

Second, given a set of input features, the choice of ranking loss used as a base for ULTR methods matters greatly (comparing the colored groups in Figure 3.3). Methods based on LambdaRank outperform methods based on the listwise softmax loss, which outperform those based on the pointwise binary cross-entropy loss. In contrast, applying ULTR techniques yields marginal ranking improvements at most compared to their naive, non-debiasing counterpart.

In detail, we observe that the pointwise two-tower and IPS models do not consistently and significantly improve performance compared to the naive pointwise loss – IPS is even significantly worse on both BERT features – and RegressionEM is the only pointwise ULTR method that brings small but sometimes significant improvements. Regarding listwise methods, IPS and DLA significantly but marginally improve on our BERT and LTR features. Finally, PairD is no better and sometimes worse than its naive LambdaRank counterpart. These results are broadly consistent across input features, suggesting that standard ULTR methods struggle to bring improvement to Baidu-ULTR, regardless of the query-document features.

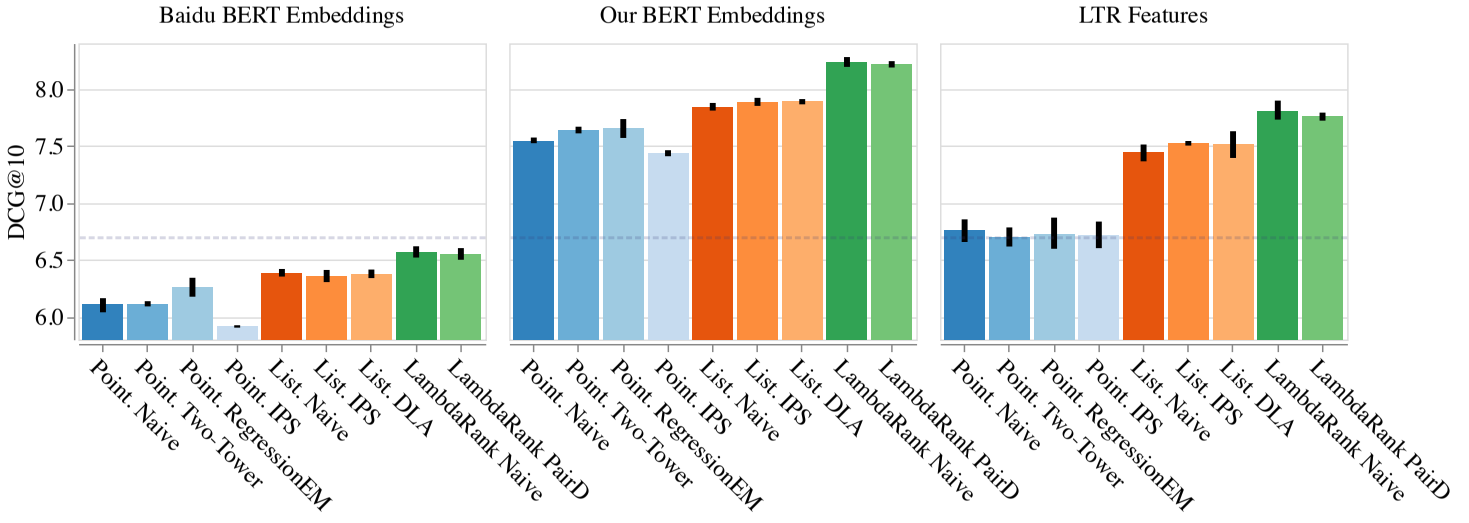


Figure 3.3: Comparing ULTR methods on pre-trained BERT embeddings and LTR features. We display the average ranking performance measured in DCG@10 over five independent runs and plot a bootstrapped 95% confidence interval. The grey dotted line indicates the performance of a random ranker.

Table 3.2: Comparison of cross-encoder models trained from scratch on the Baidu-ULTR dataset with ULTR loss functions.

Model	DCG@10 ↑	NLL ↓
Pointwise Naive	7.251	0.227
Pointwise Two-Tower	7.456	0.218
Pointwise IPS	6.296	0.222
Listwise Naive	8.478	-
Listwise IPS	7.450	-
Listwise DLA	7.802	-

Overall, we can answer research questions **RQ2.1** and **RQ2.2**: on the Baidu-ULTR dataset, unbiased learning-to-rank methods do not consistently improve ranking performance on expert annotations, particularly when contrasted with the significant and reliable differences based on the choice of query-document features and ranking loss. Our reranking results confirm the findings of Zou et al. [209].

3.6.2 Language model training is sensitive to the choice of unbiased learning-to-rank method

Given the poor performance of ULTR on our reranking datasets (see Section 3.6.1), we further investigate whether training language models with ULTR is a promising direction for future research. We train six MonoBERT [133] cross-encoders with different pointwise and listwise loss functions, including ULTR loss functions.

Table 3.2 gives an overview of applying ULTR methods during language model training. In contrast to the inefficacy of ULTR-based rerankers, we observe stark differences when applying ULTR methods during language model training. Similar to the reranking task, approaches based on the listwise softmax loss perform better than those based on pointwise binary cross-entropy. The additional application of ULTR brings considerable improvements with the pointwise two-tower objective. However, we also observe substantial degradations with both IPS and DLA. The best method is the naive listwise softmax loss without any debiasing objective.

These inconclusive results make us cautious about answering **RQ2.3**: ULTR methods substantially impact language model training more than reranking with fixed embeddings. However, the influence of ULTR on ranking performance and learned query-document representations needs further investigation.

3.6.3 Click prediction does not imply ranking performance on annotations

In addition to evaluating ranking performance, we display the negative log-likelihood (NLL) of predicted click scores in Figure 3.4. We restrict this evaluation to pointwise ULTR reranking methods, making well-defined click predictions. All ULTR methods robustly improve click prediction compared to a naive loss, with the two-tower model

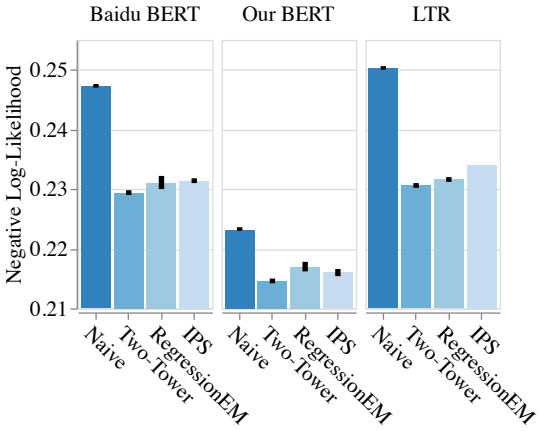


Figure 3.4: Click prediction performance of pointwise methods measured in negative log-likelihood; lower is better.

showing the largest improvement on all three datasets. Moreover, as discussed in Section 3.3.2, the propensities learned by RegressionEM are close to those discovered through intervention harvesting and show a strong position bias. This finding suggests that unbiased learning-to-rank robustly captures position bias and can better predict user clicks because of it. Yet, this improved click prediction does not translate to enhanced ranking performance on expert annotations.

Critically, we can see in Table 3.3 that ranking documents according to their (untuned) BM25 [155] scores yields better ranking performance than all click-based models we trained. Even more surprisingly, using BM25 scores as part of the input for the reranking models – it is one of the LTR features – and training these models on clicks lowers ranking performance. This phenomenon was also described by Sun et al. [166] during the WSDM Cup. These concurrent results suggest that click prediction and ranking performance on annotations are diverging objectives on Baidu-ULTR and training models on clicks does not guarantee improvements in ranking metrics.

3.7 Discussion

In light of the puzzling results reported in Section 3.6, we review potential reasons for the failure of ULTR in Section 3.7.1 as well as the limitations of our study in Section 3.7.2. Finally, we discuss the implications for the field of unbiased learning-to-rank in Section 3.7.3.

3.7.1 Potential reasons for the failure of ULTR

No position bias. A first straightforward explanation for the inefficacy of ULTR techniques would be that position bias is not prevalent in the Baidu-ULTR dataset. However, our results in Section 3.3.2 show a strongly decreasing click-through rate with

increasing position and a strong position bias estimated by techniques based on different methodologies. Recall that intervention harvesting uses CTR ratios of query-document pairs appearing in multiple positions while RegressionEM uses our BERT features and expectation maximization. The fact that these results agree strongly suggests the existence of position bias in this dataset and that ULTR methods like RegressionEM and two-tower models could capture it.

More complex user behavior. Zou et al. [209] suggest that the actual user model might be more complicated than the PBM considered in this work. This hypothesis might explain why listwise approaches bring such considerable improvements over pointwise methods and would not be surprising as user studies have identified more complex biases [2, 160, 206]. However, ULTR methods bringing no benefits would mean that the substantial position bias we identified is negligible against other types of bias. This assumption, in turn, would contradict related work where PBM-based models trained on more complex user behaviors still improved performance, albeit by not as much as the correct user model [48, 175].

Lack of identifiability. Past studies have identified that a lack of variability in log data, i.e., not encountering a query-document pair at different positions, can lead to the model not being identifiable, in the sense that there exist infinite valid combinations of relevance and bias parameters [30, 135]. However, as this work's four bias estimation methods independently converged to similar estimates, we deem this rather unlikely.

Distribution shift. Poor ranking performance may also be caused by the distribution shift between the query-document pairs in the training and annotated test set. While methods train only on the top-10 search results, the annotation dataset also includes presumably much less relevant documents sampled from the top-1000 candidate documents. This shift in the input feature distribution, in conjunction with the low overlap between train and test queries (see Section 3.3.1) and the long-tail query distribution during training, might all potentially impact ULTR. More research and, ideally, clicks for queries with expert annotations are needed to better understand the impact of the various distribution shifts on ULTR.

Strong logging policy. While we do not have access to logging policy scores and rankings on the annotated test queries, the limitations imposed by an already strong production system have been well-documented [47]. A side observation of ours reinforces this possibility: methods that show strong ranking performance correlate highly with the logging policy of the click dataset (as estimated through the original item position). For future dataset releases, we recommend including logging policy scores to enable assessments of the improvement made by training new rankers on click data compared to the logging policy [47, 75].

User-annotator disagreement. Finally, the divergence of click prediction and ranking performance hints at a deeper issue: annotators may not find the same documents relevant as users scrolling through a result page, which might be personalized and part

of a multi-query session to find actionable information for the current user needs. We hope further analysis of Baidu-ULTR or parallel datasets of clicks and expert annotations can clarify this hypothesis.

3.7.2 Limitations

First, we only considered the correction of position bias under the position-based model, while other biases might have to be mitigated [2, 175]. Second, we did not include bias features available in Baidu-ULTR beyond positions. We highlight that Chen et al. [33] report improved ranking performance by considering additional bias features, including item height or media type. Third, the variability in item position that we used for position bias estimation is likely not due to natural variability but due to unobserved confounding and algorithmic choices based on additional context information rather than a stochastic logging policy. Fourth, our work only applied ULTR during language model pre-training and on fixed BERT embeddings for reranking. We did not yet explore multi-stage pre-training schemes, such as pre-training BERT on a naive click prediction task and later fine-tuning BERT using ULTR.

Lastly, we stress that our findings are strictly limited to the Baidu-ULTR dataset. While the dataset was collected from one of the most used search engines globally, it cannot be considered representative of all real-world search scenarios. Yet, we believe that results on this large-scale dataset are important for the research community.

3.7.3 Implications for the field

Our results starkly contrast with many studies in semi-synthetic simulation [5, 7, 90, 99, 157, 173] and call for adjusting semi-synthetic experimental setups to reflect real-world challenges better. The richness of this dataset allows for seeding simulations with more plausible data and, therefore, might bridge the gap between simulations and reality. Moreover, the prevalence of transformer-based models and text embeddings in current IR research encourages exploring the interaction of ULTR methods with such models. As shown in Section 3.6.2, the same techniques applied during language model training, instead of on a small re-ranking model, can yield vastly different results.

More broadly, the puzzling results we obtained, especially the apparent divergence between clicks and relevance annotations, prompt us to rethink how we measure success in ULTR. Expert annotations are static and might not reflect user context, so results obtained on annotated datasets can be polluted by distribution shifts or logging policy performance. In particular, we believe that, whenever possible, we should evaluate ULTR methods on the tasks they are trained to accomplish, including relevance estimation, bias estimation, CTR maximization, fairness of exposure, etc.

Finally, we would like to stress that this chapter is not a judgment on the ULTR field. As mentioned above, results may differ on other datasets, and most ULTR methods are theoretically justified, meaning only their validity in this scenario has been challenged. In fact, our experiments validate some intuitions formulated in the ULTR community. For instance, the stark differences between loss functions justify the motivation of Joachims et al. [99] to connect simple user models with more powerful ranking loss functions.

Table 3.3: Comparison of ULTR methods on embeddings obtained from the pre-trained MonoBERT released by Baidu [209]. We display averaged results over five independent runs with standard deviation in parentheses. ↑ indicates the higher the better and ↓ the lower the better. Methods are grouped by loss, and significant differences are marked with ▲ or ▼ compared to the naive method in each group using a two-sided paired t-test with $\alpha = 0.01$ and Bonferroni correction.

Baselines						
Method	DCG@1 ↑	DCG@3 ↑	DCG@5 ↑	DCG@10 ↑	MRR@10 ↑	NLL ↓
Random	1.472 (0.020)	3.148 (0.030)	4.349 (0.036)	6.693 (0.044)	0.577 (0.003)	0.944 (0.002)
BM25 ($k_1 = 1.2, b = 0.75$)	2.211	4.656	6.377	9.544	0.716	-
Baidu MonoBERT						
Method	DCG@1 ↑	DCG@3 ↑	DCG@5 ↑	DCG@10 ↑	MRR@10 ↑	NLL ↓
Pointwise Naive	1.264 (0.030)	2.765 (0.059)	3.881 (0.068)	6.111 (0.079)	0.535 (0.007)	0.246 (0.005)
Pointwise Two-Tower	1.266 (0.018)	2.769 (0.023)	3.893 (0.026)	6.114 (0.031)	0.533 (0.003)	0.228 (0.005)▼
Pointwise RegressionEM	1.343 (0.034)▲	2.884 (0.074)▲	4.017 (0.091)▲	6.257 (0.107)▲	0.548 (0.005)▲	0.229 (0.005)▼
Pointwise IPS	1.157 (0.011)▼	2.606 (0.012)▼	3.699 (0.013)▼	5.915 (0.014)▼	0.517 (0.002)▼	0.230 (0.005)▼
Listwise Naive	1.362 (0.022)	2.940 (0.037)	4.096 (0.037)	6.388 (0.042)	0.549 (0.004)	-
Listwise IPS	1.353 (0.034)	2.920 (0.060)	4.078 (0.063)	6.359 (0.071)▼	0.548 (0.005)	-
Listwise DLA	1.345 (0.020)	2.917 (0.037)	4.083 (0.048)	6.378 (0.051)	0.549 (0.004)	-
LambdaRank Naive	1.419 (0.017)	3.047 (0.039)	4.237 (0.054)	6.570 (0.065)	0.562 (0.005)	-
LambdaRank PairD	1.410 (0.028)	3.025 (0.044)	4.215 (0.064)	6.550 (0.066)	0.559 (0.005)	-

Table 3.4: Comparison of ULTR methods on embeddings obtained from our pre-trained MonoBERT model, displaying averaged results over five independent runs with standard deviation in parentheses. ↑ indicates the higher the better and ↓ the lower the better. Methods are grouped by loss, and significant differences are marked with ▲ or ▼ compared to the naive method in each group using a two-sided paired t-test with $\alpha = 0.01$ and Bonferroni correction.

Our MonoBERT						
Method	DCG@1 ↑	DCG@3 ↑	DCG@5 ↑	DCG@10 ↑	MRR@10 ↑	NLL ↓
Pointwise Naive	1.705 (0.013)	3.602 (0.022)	4.944 (0.023)	7.546 (0.031)	0.619 (0.002)	0.221 (0.005)
Pointwise Two-Tower	1.656 (0.017)▼	3.556 (0.024)▼	4.948 (0.032)	7.639 (0.038)▲	0.615 (0.004)▼	0.213 (0.006)▼
Pointwise RegressionEM	1.694 (0.031)	3.619 (0.056)	4.998 (0.081)▲	7.657 (0.111)▲	0.618 (0.004)	0.215 (0.005)
Pointwise IPS	1.589 (0.012)▼	3.438 (0.012)▼	4.794 (0.013)▼	7.436 (0.035)▼	0.604 (0.001)▼	0.214 (0.005)
Listwise Naive	1.798 (0.008)	3.768 (0.020)	5.167 (0.029)	7.844 (0.046)	0.631 (0.001)	-
Listwise IPS	1.816 (0.017)	3.800 (0.027)▲	5.200 (0.031)▲	7.885 (0.043)▲	0.634 (0.002)▲	-
Listwise DLA	1.816 (0.011)	3.806 (0.022)▲	5.210 (0.024)▲	7.890 (0.031)▲	0.634 (0.001)▲	-
LambdaRank Naive	1.931 (0.016)	3.993 (0.029)	5.452 (0.040)	8.233 (0.055)	0.648 (0.003)	-
LambdaRank PairD	1.932 (0.007)	3.985 (0.019)	5.438 (0.022)	8.216 (0.038)	0.646 (0.003)	-

Table 3.5: Comparison of ULTR methods on common numerical learning-to-rank features, displaying averaged results over five independent runs with standard deviation in parentheses. ↑ indicates the higher the better and ↓ the lower the better. Methods are grouped by loss, and significant differences are marked with ▲ or ▼ compared to the naive method in each group using a two-sided paired t-test with $\alpha = 0.01$ and Bonferroni correction.

LTR Features						
Method	DCG@1 ↑	DCG@3 ↑	DCG@5 ↑	DCG@10 ↑	MRR@10 ↑	NLL ↓
Pointwise Naive	1.312 (0.034)	2.977 (0.066)	4.246 (0.090)	6.757 (0.133)	0.543 (0.008)	0.247 (0.005)
Pointwise Two-Tower	1.333 (0.046)	2.986 (0.085)	4.241 (0.094)	6.698 (0.110)▼	0.553 (0.008)▲	0.228 (0.006)▼
Pointwise RegressionEM	1.397 (0.068)▲	3.062 (0.133)▲	4.300 (0.166)▲	6.729 (0.181)	0.559 (0.014)▲	0.229 (0.005)▼
Pointwise IPS	1.352 (0.044)▲	3.013 (0.095)	4.258 (0.126)	6.717 (0.154)	0.560 (0.011)▲	0.232 (0.005)▼
Listwise Naive	1.599 (0.023)	3.485 (0.042)	4.845 (0.072)	7.443 (0.100)	0.596 (0.005)	-
Listwise IPS	1.641 (0.020)▲	3.545 (0.012)▲	4.920 (0.020)▲	7.522 (0.026)▲	0.602 (0.002)▲	-
Listwise DLA	1.621 (0.026)	3.525 (0.074)▲	4.891 (0.101)▲	7.512 (0.160)▲	0.599 (0.006)▲	-
LambdaRank Naive	1.750 (0.035)	3.717 (0.068)	5.125 (0.083)	7.810 (0.111)	0.613 (0.007)	-
LambdaRank PairD	1.723 (0.018)▼	3.683 (0.034)▼	5.089 (0.042)▼	7.761 (0.046)▼	0.608 (0.004)▼	-

3.8 Conclusion

In this chapter, we investigated **RQ2** on whether established unbiased learning-to-rank methods for position bias correction improve ranking performance by carefully revisiting and extending the experiments by Zou et al. [209] on the Baidu-ULTR dataset. As the largest publicly available dataset comprising both click logs from a major search engine and expert annotations, this dataset constitutes a rare opportunity to assess the progress of ranking methods using implicit feedback, and especially unbiased learning-to-rank.

We conclude that prevalent ULTR techniques do not yield clear improvements in ranking performance compared to their naive, non-debiasing counterparts on the Baidu-ULTR dataset (**RQ2.1**), especially given the substantial improvements we observe in the choice of query-document representations and ranking loss (**RQ2.2**). Critically, our results demonstrate that applying ULTR methods as black-box systems does not always improve ranking performance and can even degrade it, as we find when applying IPS-based methods during transformer pre-training (**RQ2.3**). Our findings confirm the work by Zou et al. [209], despite using different dataset features and preprocessing, model implementations, and performed extensive hyperparameter tuning.

While we find that ULTR methods achieve robust improvements in click prediction performance in this chapter, gains in click prediction do not always translate into increased ranking performance on expert annotations. We further observed a divergence between click-based and annotation-based objectives, as all click-based approaches were outperformed on annotation-based metrics by simple baselines such as BM25, even when BM25 was used as an input to a ranking model trained on clicks.

The findings of this chapter call for more research into the conditions for real-world success and failure of unbiased learning-to-rank and click-based approaches as a whole, which we hope to facilitate with the datasets, pre-trained models, and implementations created in this chapter. In the next chapter, we conduct one such investigation by focusing specifically on two-tower models, a widely adopted neural click model architecture in industry [72, 85, 105], to understand better how and when these models work.

3.9 Reproducibility

In the following, we link all resources created in this chapter.

3.9.1 Datasets

We release two pre-processed subsets of the Baidu-ULTR dataset for reranking experiments, each containing 768-dimensional MonoBERT embeddings and traditional learning-to-rank features (BM25, TF-IDF, query-likelihood, etc.). The first dataset uses embeddings from the original MonoBERT cross-encoder released by Zou et al. [209], while the second uses embeddings from our independently trained MonoBERT model:

- Baidu-ULTR reranking dataset with Baidu MonoBERT embeddings:
  https://huggingface.co/datasets/philipphager/baidu-ultr_baidu-mlm-ctr

- Baidu-ULTR reranking dataset with our MonoBERT embeddings:
  https://huggingface.co/datasets/philipphager/baidu-ultr_uva-mlm-ctr

3.9.2 Code and models

We provide JAX implementations of all ULTR methods trained and evaluated in this chapter. Additionally, we release the code and weights of all six MonoBERT cross-encoders trained from scratch in this work. Lastly, we also release the code for three established methods for position-bias estimation in an offline setting:

- Main repository containing all code to train, tune, and evaluate all reranking methods, including reference implementations of six ULTR methods in JAX:
  https://github.com/philipphager/ultr-reproducibility
- JAX code for distributed MonoBERT training:
  https://github.com/philipphager/baidu-bert-model
- Code for three intervention harvesting methods for position-bias estimation:
  https://github.com/philipphager/ultr-bias-toolkit

4

Unidentified and Confounded? Understanding Two-Tower Models for Unbiased Learning to Rank

After our empirical comparison of ULTR methods in Chapter 3, we focus on two-tower models, a neural click model architecture that is widely used for bias correction in industry [72, 85, 105, 195, 202]. Following the position-based model, two-tower models assume that users click only on items they observe and find relevant [195]. This assumption is implemented through two neural networks: a *bias tower* predicts document observation from contextual features, while a *relevance tower* predicts document relevance from content features.[1] During training, both towers jointly predict clicks, while at inference, only the relevance tower is used to rank documents. Two-tower models are easy to implement, scale, and extend to bias features beyond position, e.g., to learn different examination patterns across mobile and desktop devices [33, 195].

Recent work makes a surprising observation: the ranking performance of two-tower models deteriorates in simulation when training on clicks collected by a strong production system that already places relevant documents in top positions [200]. Existing work proposes two explanations. First, strong production systems may act as confounders by correlating position and relevance features [122, 200]. Second, identifiability issues may arise: factorizing clicks into bias and relevance typically requires observing documents across positions (document swaps) [3, 30, 39], and production systems might not collect enough swaps to recover model parameters from clicks accurately. This finding is concerning because it suggests practitioners face a tension between improving their production systems and training unbiased ranking models. Therefore, we investigate:

RQ3 When can two-tower models reliably disentangle bias and relevance signals from clicks, and what role do the production systems play collecting the clicks?

To answer this question, we first investigate identifiability conditions for recovering bias and relevance parameters from observed clicks. We then analyze how logging policies

This chapter was published as P. Hager, O. Zoeter, and M. de Rijke. Unidentified and confounded? Understanding two-tower models for unbiased learning to rank. In *Proceedings of the 2025 International ACM SIGIR Conference on Innovative Concepts and Theories in Information Retrieval*, 2025.

[1] Note that the neural PBM in Chapter 2 multiplicatively combines bias and relevance probabilities, while this chapter uses an additive combination of log-odds. Yan et al. [195] discuss common tower combinations.

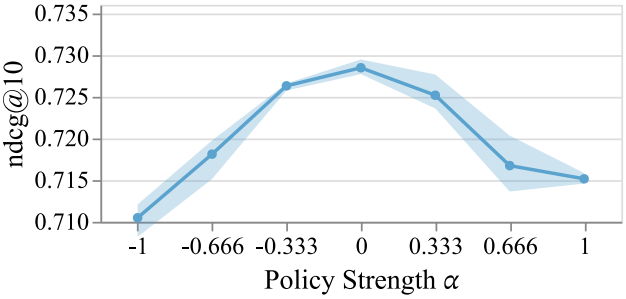


Figure 4.1: Two-tower models trained on deterministic logging policies of varying strengths (α) on MSLR30K: $\alpha = 1$ represents sorting by expert annotations, $\alpha = 0$ random sorting, and $\alpha = -1$ inversely ranking from least to most relevant.

affect parameter estimation in two-tower models. And lastly, we validate our theory in semi-synthetic simulations and discuss their practical implications.

4.1 Introduction

Unbiased learning-to-rank (ULTR) addresses the challenge of optimizing ranking models for search and recommendation using implicit user feedback [76, 99, 183]. Clicks, in particular, are widely used for training models, yet they often provide a biased signal of user preferences and should not be naively interpreted as positive or negative preference [39, 97]. In web search, for example, clicks can be influenced by a document's position on the result page [97], its surrounding items [39, 192, 206], or the user's trust in the search engine to list relevant items first [2, 97, 176].

Additive two-tower models have emerged as a popular solution for addressing click biases in industrial settings due to their flexibility and scalability [72, 85, 105, 195, 202]. These models consist of two neural networks, referred to as towers: the first network, or relevance tower, predicts item relevance based on content-related features. The second network, or bias tower, models user examination biases from contextual features such as document position, device type (e.g., mobile vs. desktop), or display height on the screen [33, 195]. During training, outputs from both towers are combined to predict user clicks, whereas, at serving time, only the relevance tower is used. The bias tower effectively acts as a mechanism for debiasing the relevance tower during training.

Recent studies describe an interesting phenomenon when using two-tower models. As production logging policies become more effective at surfacing relevant items, the ranking performance of two-tower models trained on clicks collected by those policies steadily declines [200]. This is a troubling finding because most companies seek to optimize their production systems for ever-better ranking performance. Two key concepts have been put forward as contributing factors: *confounding* and *identifiability*. Zhang et al. [200] theorize that stronger logging policies placing more relevant items in top positions act as a confounder by introducing a correlation between bias features and document relevance that two-tower models cannot disentangle. Likewise, position

bias estimations of two-tower models might be biased on data collected by strong logging policies [122]. Although an explanation in terms of confounding is intuitive, it is unsatisfactory as two-tower models take both relevance-related document features and position into account.[2] Yet, the empirical results seem consistent, and we observe the same effect in preliminary reproducibility experiments. Figure 4.1 displays the ranking performance of two-tower models trained on data collected by policies of varying strengths;[3] ranking performance decreases when the logging policy is not uniformly random, indicating the existence of an underlying mechanism.

Chen et al. [30] discuss an alternative explanation in terms of identifiability. Intuitively, identifiability asks under which conditions we can (uniquely) recover model parameters from observed data. Inferring model parameters in click modeling is a well-known challenge [37, 99]. E.g., estimating position bias parameters typically requires observations of the same query-document pair across two positions (a swap) to correctly attribute changes in clicks to bias rather than a shift in document relevance [3, 39, 60]. Chen et al. [30] analyze click models following the Examination Hypothesis (the assumption that clicks factorize into bias and relevance terms, which two-tower models make [195]) and show that we can only recover click model parameters on data in which all positions are *connected* through document swaps. The influence of logging policies on identifiability seems clear: our production system might not collect suitable data for training two-tower models, e.g., by never swapping documents across positions [30]. The experiments on logging policy confounding in [122, 200] and Figure 4.1 use deterministic logging policies without document swaps, suggesting identifiability may be a culprit. However, work on identifiability suggests rather binary observations, i.e., sufficient swaps connect all positions or they do not. [30]. Hence, can identifiability explain a gradually deteriorating ranking performance? And is there still an effect of logging policies on two-tower models if identifiability is guaranteed?

Given their widespread usage in industry and academia, it is important to understand precisely when two-tower models work and under what conditions they might fail. Thus, this chapter tackles the following three research questions:

RQ3.1 What are the conditions for identifying two-tower models?

RQ3.2 Do logging policies impact two-tower models beyond identifiability?

RQ3.3 What are strategies to alleviate potential logging policy influences?

We first show that two-tower models require either randomized document swaps or overlapping feature distributions across ranks to be identified (**RQ3.1**). We then show that while logging policies do not influence well-specified two-tower models, they can amplify biases in misspecified models when prediction errors correlate with document placement (**RQ3.2**). Finally, we propose a sample weighting technique that helps to alleviate the bias amplification introduced by logging policies (**RQ3.3**).

[2] Two-tower models condition on relevance features and position, essentially already blocking the problematic paths highlighted in [200, Figure 2b] and [122, Figure 1b].

[3] Different strengths in Figure 4.1 are based on sorting by expert relevance annotations with additive noise [47, 200]. Section 4.5.3 shows that this policy simulation is problematic.

Our contributions are:

- A theoretical analysis disentangling the problem of identification and logging policy effects for additive two-tower models.
- A propensity weighting scheme helping to alleviate logging policy effects on misspecified two-tower models.
- A thorough empirical evaluation of all theoretical contributions through simulation experiments.
- A discussion of practical takeaways for researchers and industry practitioners working with two-tower models.

4.2 Related Work

4.2.1 Two-tower models for unbiased learning-to-rank

Guo et al. [72] introduce the industry practice of using two-tower models to account for click biases to the research community. Yan et al. [195] closely examine and question the additive assumption in two-tower models and suggest alternatives that account for mixtures of user behaviors. Zhao et al. [202] train two-tower models with multiple types of feedback beyond clicks for video recommendation at YouTube. Haldar et al. [85] and Khrylchenko and Fritzler [105] use two-tower models for search at Airbnb and Yandex respectively. Zhuang et al. [206] and Wu et al. [192] include surrounding items and user context when modeling bias towers. Hager et al. [79] investigate parallels between two-tower models and pointwise inverse propensity scoring [99, 137, 176]. And Chen et al. [33] show the importance of considering bias features beyond position in search.

4.2.2 Logging policy confounding

The idea that logging policy performance affects two-tower models and click models more broadly has recently gained attention in unbiased learning-to-rank [47, 48, 122, 200]. Deffayet et al. [48] show that a strong ranking performance of click models does not guarantee inferring unbiased parameters when training on strong logging policies. Zhang et al. [200] introduce logging policy confounding for additive two-tower models, arguing that policies performing better than a random ranker break the independence assumption between bias and relevance towers. They use deterministic logging policies derived from expert annotations, which likely leads to identifiability problems (cf. Section 4.4) and their logging policy simulation likely introduces new confounding problems (cf. Section 4.5.3). We build on their findings, separating logging policy effects from identifiability problems, confirming their empirical results, but showing that the primary problems are model misspecification and a flawed simulation setup. Luo et al. [122] find that positions containing more relevant documents systematically inflate examination probabilities. Empirically, we confirm their findings under certain conditions and propose a simple sample-weighting scheme for binary cross-entropy to adjust for unequal item exposure under a logging policy reminiscent of corrections used in intervention harvesting for position bias estimation [3, 14, 60].

4.2.3 Identifiability

The study of identifiability, fundamentally the question about when we can recover model parameters from observational data, dates back decades [92, 111, 112], but only recently emerged in unbiased learning-to-rank. Oosterhuis [135] demonstrates that a lack of document swaps across positions leads to cases in which infinitely many model parameters explain observed clicks equally well, leading to inconsistent click models. Chen et al. [30] connected this example to formal identifiability theory, finding that click models following the Examination Hypothesis [39, 153] require randomized swaps creating a connected position graph. Our work extends this analysis to additive two-tower models and shows that identifiability can be achieved not only through exact document swaps but also through overlapping feature distributions. While Chen et al. [30] noted logging policies might affect identifiability, we demonstrate they can impact two-tower models in ways that extend beyond identifiability issues in practical settings.

4.3 Two-Tower Models

Our analysis focuses on additive two-tower models, the most prevalent type in practical applications [72, 85, 195, 202]. Each tower processes different inputs: one tower handles relevance-related query-document features, while the other uses context-dependent bias features. We will focus the majority of our analysis on position bias [39, 97]. Each tower outputs logits[4] which are added together and transformed to a click probability via the sigmoid function:

$$P(C = 1 \mid q, d, k) = \sigma(\theta_k + \gamma_{q,d}), \text{with: } \sigma(x) = (1 + e^{-x})^{-1}. \tag{4.1}$$

where the probability of click C is a combination of a position bias logit $\theta_k \in \mathbb{R}$ for rank $k \in K, K = \{1, 2, \dots\}$, and a relevance logit $\gamma_{q,d} \in \mathbb{R}$ for document $d \in D$ and query $q \in Q$. We use $\sigma(\cdot)$ to denote the sigmoid function. The most common way to train this model is by minimizing the expected negative log-likelihood:

$$\begin{aligned}\mathcal{L}(\theta, \gamma) = -\mathbb{E}_{(q,d,k,c)\sim P(Q,D,K,C)}\big[c \log \sigma(\theta_k + \gamma_{q,d}) \\ +(1-c)\log(1 - \sigma(\theta_k + \gamma_{q,d}))\big].\end{aligned} \tag{4.2}$$

As the true data-generating distribution $P(Q, D, K, C)$ is unknown, we typically estimate model parameters by minimizing the empirical negative log-likelihood [195]:

$$\begin{aligned}\hat{\mathcal{L}}(\theta, \gamma : D) = -\frac{1}{N} \sum_{(q,d,k,c)\in\mathcal{D}} \big[c \log \sigma(\theta_k + \gamma_{q,d}) \\ +(1-c)\log(1 - \sigma(\theta_k + \gamma_{q,d}))\big],\end{aligned} \tag{4.3}$$

over a dataset of N observations: $\mathcal{D} = \{(q_i, d_i, k_i, c_i)\}_{i=1}^{N}$.

[4] Logits are the natural logarithm of the odds of a probability p: $\text{logit}(p) = \ln(\text{odds}(p))$.

4.4 Identifiability of Two-Tower Models

We begin our analysis by examining **RQ3.1** on when we can uniquely recover the parameters of two-tower models from click data, fundamentally a question of identifiability. We will define identifiability, analyze it for additive two-tower models similar to Chen et al. [30], and extend this analysis to identifiability from query-document features.

4.4.1 Identifiability

We define identifiability following the standard work of Koller and Friedman [109, Definition 19.4], whereby a model is identifiable if no two distinct sets of parameters $(\theta_k, \gamma_{q,d}), (\theta'_k, \gamma'_{q,d})$ induce the same distribution over observable variables. That is, if:

$$P(C \mid q, d, k; \theta, \gamma) = P(C \mid q, d, k; \theta', \gamma') \tag{4.4}$$

it must follow that $\theta_k = \theta'_k$ for all positions and $\gamma_{q,d} = \gamma'_{q,d}$ for all query-document pairs. This is an important foundational property in probabilistic modeling ensuring a one-to-one (injective) mapping between model parameters and the observable data distribution. Unidentifiability, in turn, implies that we cannot uniquely recover the true model parameters from observed data as there are multiple (possibly infinite) parameter combinations that explain the observed data equally well.

4.4.2 Invariance to parameter shift

To begin our analysis, we first must acknowledge that two-tower models face an inherent identifiability challenge: we only observe clicks, while position bias and document relevance remain unobserved. This leads to a fundamental problem of parameter shifts: we can arbitrarily increase our relevance parameters while decreasing our bias parameters by the same amount, resulting in identical click probabilities but different parameter values.

To demonstrate this shift, let's consider a set of alternative parameters θ'_k and $\gamma'_{q,d}$ next to our true model parameters θ_k and $\gamma_{q,d}$. Since the sigmoid function $\sigma(\cdot)$ is injective, identical click probabilities require identical inputs to the sigmoid. Therefore, any two parameter sets producing the same click probability must satisfy:

$$\theta_k + \gamma_{q,d} = \theta'_k + \gamma'_{q,d} \quad \forall\,(q, d, k). \tag{4.5}$$

We can construct valid alternative parameters that follow Eq. 4.5 by considering a real-valued constant $\Delta_k \in \mathbb{R}$ for any rank $k \in K$:

$$\theta'_k = \theta_k + \Delta_k, \quad \gamma'_{q,d} = \gamma_{q,d} - \Delta_k. \tag{4.6}$$

This shows the invariance of additive two-tower models to additive parameter shifts. By rearranging, we highlight:

$$\gamma_{q,d} - \gamma'_{q,d} = \theta'_k - \theta_k = \Delta_k, \tag{4.7}$$

that the left-hand side depends solely on query q and document d, while the right-hand side depends only on the rank k. Consequently, when documents appear exclusively in

a single position (never appearing in multiple ranks), we can choose an arbitrary offset Δ_k to shift their relevance parameters and adjust the bias parameter accordingly without changing the overall click probabilities. To illustrate, consider two documents d_1 and d_2 that are displayed exclusively at ranks 1 and 2 respectively:

$$\gamma_{q,d_1} - \gamma'_{q,d_1} = \Delta_1, \quad \gamma_{q,d_2} - \gamma'_{q,d_2} = \Delta_2. \tag{4.8}$$

Without observing one document across both positions, the offsets Δ_1 and Δ_2 can be chosen independently, preventing a unique decomposition of bias and relevance parameters. Therefore, *additive two-tower models are unidentifiable when observing each query-document pair only at a single position [30, 135].*

4.4.3 Identification through randomization

Now suppose that we observe a query-document pair q, d across positions 1 and 2. Note that the left-hand side of Eq. 4.7 depends only on the current document. Thus, observing the exact same query-document pair across two positions forces the parameter offsets between both positions to be equal:

$$\gamma_{q,d} - \gamma'_{q,d} = \Delta_1 = \Delta_2. \tag{4.9}$$

Note that we do not need all query-document pairs to swap across all positions. Chen et al. [30] observe that it is sufficient that an undirected graph $G = (V, E)$, where vertices V correspond to positions and edges E exist between each pair of ranks that shares at least one query-document pair, is connected [30, Theorem 1]. Similarly, for additive two-tower models, if a graph of positions is connected, all rank-dependent offsets Δ_k must be equal to a global offset Δ.[5] To readers familiar with literature on position bias estimation, it might be useful to think of different randomization schemes as different ways to connect a graph of positions [39, 99, 148]. Note that when the graph contains disconnected components, each component can have its own arbitrary parameter shift, independent of other components, which fundamentally prevents the recovery of a single unique set of model parameters.

Thus, random swaps unify parameter shifts across positions to a single global parameter shift. This last remaining ambiguity is commonly resolved through normalizing parameters [112, Section 6.3]. A standard normalization for additive parameter shifts are location normalizations, e.g., fixing the first bias parameter $\theta_1 = 0$ (used in this work) or enforcing bias parameters to be centered around zero: $\sum_{k=1}^{K} \theta_k = 0$. By normalizing bias parameters, we can finally recover a unique set of both bias and relevance parameters. Thus, *additive two-tower models are identifiable up to an additive constant when observing query-document pairs across positions, such that all positions form a connected graph.*

4.4.4 Identification through overlapping features

In most applications of two-tower models, we do not assign independent model parameters to each query-document pair. Instead, we predict relevance from a shared feature

[5] Chen et al. [30] consider multiplicative parameter shifts under the Examination Hypothesis [39, 153], but we can directly apply their graph reasoning to additive shifts.

representation. In this setting, every query-document pair is represented by a feature vector $x_{q,d} \in \mathbb{R}^m$ and the relevance tower $r(\cdot)$ consists, for example, of a linear model or a neural network:

$$P(C = 1 \mid q, d, k) = \sigma(\theta_k + r(x_{q,d})). \tag{4.10}$$

Note that when discussing neural networks in this chapter, we ask if the output logits are identified (nonparametric identification [112, Section 6]) not the network's parameters. Our main result builds on [30, Theorem 1] and shows that identifiability is possible without swaps as long as query-document features overlap across ranks:

Theorem 2 (Identifiability through feature overlap)**.** *Let $G = (V, E)$ be an undirected graph where vertices V correspond to positions and an edge $(k, k') \in E$ exists if the feature support[6] between positions k and k' overlap:*

$$\textit{supp}\,(P(x \mid k)) \cap \textit{supp}\,(P(x \mid k')) \neq \emptyset. \tag{4.11}$$

If G is connected and the relevance tower $r(\cdot)$ is continuous, then the additive two-tower model is approximately identifiable up to an additive constant.

Proof. Consider the alternative parameterization θ'_k and $r'(\cdot)$ that yield identical click probabilities to our original parameters θ_k and $r(\cdot)$. Recalling Eq. 4.7, for these parameterizations to produce the same click probabilities, the following must hold:

$$r(x_{q,d}) - r'(x_{q,d}) = \theta'_k - \theta_k = \Delta_k, \tag{4.12}$$

where Δ_k is a rank-dependent offset. For any two positions k and k' that share overlapping feature support, there exist query-document pairs with feature vectors $x_1 \sim P(x|k)$ and $x_2 \sim P(x|k')$ such that $x_1 \approx x_2$. By the continuity of $r(\cdot)$ and $r'(\cdot)$, we have:

$$r(x_1) - r'(x_1) = \Delta_k \quad \text{and} \quad r(x_2) - r'(x_2) = \Delta_{k'}. \tag{4.13}$$

Assuming that $r(\cdot)$ and $r'(\cdot)$ share a Lipschitz constant L, we can bound the difference in parameter offsets between positions as:

$$|\Delta_k - \Delta_{k'}| \leq 2L \cdot \mid x_1 - x_2 \mid_2 . \tag{4.14}$$

This bound shows that when feature vectors are similar between positions, their parameter offsets must also be similar. As the feature overlap increases (i.e., as $\mid x_1 - x_2 \mid_2 \to 0$), the difference in offsets approaches zero. Given that the graph G is connected, there exists a path between any two positions k and k'. Along this path, the parameter offsets between adjacent positions are approximately equal due to the continuity constraint. By transitivity across the connected graph, all position-dependent offsets Δ_k must converge to a single global offset Δ up to an approximation error that diminishes as feature overlap increases. After normalization (e.g., setting $\theta_1 = 0$), the model parameters are uniquely identified. □

[6]The support $\text{supp}\,(P(x \mid k)) = \{x : P(x \mid k) > 0\}$ is the region of the feature space containing all possible query-document feature vectors with a non-zero probability of appearing at rank k.

We showed that when query-document features are similar between positions, their parameter offsets must also be similar. Therefore, to identify a model from features, we need feature spaces that overlap between positions. The overlap condition ensures that we can find similar query-document features across ranks and the continuity condition constrains parameter offsets across positions to enable approximate model identification. Thus, *when document swaps across ranks are unavailable, a continuous relevance model combined with overlap in the feature distributions across positions can (approximately) identify two-tower models up to an additive constant*.

In practice, many two-tower models use bias features beyond position, such as device platform, content type, or the display height on screen [33, 195, 209]. Such information might be captured in a feature vector $z_{q,d} \in \mathbb{R}^n$ serving as input to the bias tower $b(\cdot)$:

$$P(C = 1 \mid q, d) = \sigma(b(z_{q,d}) + r(x_{q,d})). \tag{4.15}$$

Our identifiability Theorem 2 directly extends to this model setup, but with increased combinatorial complexity. The model remains identifiable only when sufficient overlap exists in document distributions across the entire space of bias feature combinations. We direct the interested reader on this topic to Appendix 4.A.

4.4.5 Practical pitfalls of overlapping features

The previous section makes two key assumptions for identifiability that can be difficult in practice: overlapping document features and a continuous relevance model. First, feature overlap decreases with increasing dimensionality [40]. That is, as we add more query-document features, documents become less likely to be sufficiently close in feature space. Therefore, we should aim to use fewer query-document features, use dimensionality reduction methods, or introduce document swaps to guarantee overlapping support.

Secondly, the continuity assumption requires that small differences in feature space do not cause large, discontinuous jumps in relevance predictions. While neural networks are continuous, deep networks can produce large jumps in relevance prediction even for minor feature differences (i.e., unregularized deep neural networks can have large Lipschitz constants [161]). Thus, when randomized data is unavailable, it is advisable to use shallow neural networks, regularization, or making parametric assumptions when possible to limit the expressiveness of the relevance model. Note that this advice may conflict with our later discussion of model misspecification in Section 4.5.2.

4.5 Influence of the Logging Policy

We have seen that identifying additive two-tower models requires overlap in distributions across ranks, either through explicit randomization or shared query-document features that allow the disentangling of bias and relevance. Therefore, it is important first to highlight that logging policies govern the query-document distributions across ranks and, thus, have a major impact on identifiability. E.g., a single, deterministic logging policy will never collect a dataset that is suitable to identify a two-tower model with separate relevance parameters per query-document pair, as this model requires explicit

document swaps (Section 4.4.3). Even when we generalize across query-document features, we require overlap in the feature distributions across ranks, which are fundamentally governed by the logging policy. Thus, *logging policies may collect data that is insufficient for identifying two-tower models.*

4.5.1 Influence beyond identifiability?

The connection between a logging policy and identifiability seems clear: we must collect overlapping feature or document distributions across ranks. Our analysis so far, and prior work on identifiability [30], suggest that ensuring that a logging policy collects sufficient document swaps across all bias dimensions guarantees that two-tower models can recover model parameters given enough data. Recent work [122, 200] and our experiment in Figure 4.1, however, found gradual degradation in ranking performance when training two-tower models on logging policies with gradually increasing ranking performance, hinting at an influence of the logging policy beyond a rather binary identifiability condition. Next, we investigate the role logging policies might play beyond identifiability concerns when training two-tower models (**RQ3.2**).

4.5.2 Logging policy impact on model estimation

For simplicity, we examine the role of the logging policy for a two-tower model with separate relevance parameters for each query-document pair. Our findings below translate to the case in which we generalize over query-document features. To begin, we highlight the role of the logging policy by rewriting the expected negative log-likelihood from Eq. 4.2:

$$\begin{aligned}\mathcal{L}(\theta, \gamma) = -\sum_{q} P(q) \sum_{d,k} \pi(d, k \mid q) \big[P(C=1 \mid q, d, k) \cdot \\ \log \sigma(\theta_k + \gamma_{q,d}) + P(C=0 | q, d, k) \log(1 - \sigma(\theta_k + \gamma_{q,d})) \big].\end{aligned} \tag{4.16}$$

We define a policy as the joint probability $\pi(d, k \mid q)$ of a document d being shown at rank k for query q. We can observe that the policy weights the contribution of each query-document pair to the loss. The analysis below separates the impact of logging policies on well-specified versus misspecified models.

Lemma 1 (No policy impact on well-specified models)**.** *If a two-tower model is well-specified (i.e., can perfectly model the true click probabilities) and identifiable, then the logging policy has no effect on the estimated model parameters.*

Proof. The partial derivatives of the loss with respect to the model parameters are:

$$\frac{\partial \mathcal{L}}{\partial \gamma_{q,d}} = P(q) \sum_{k} \pi(d, k \mid q) \big[P(C=1 \mid q, d, k) - \sigma(\theta_k + \gamma_{q,d}) \big] = 0, \tag{4.17}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_k} = \sum_{q} P(q) \sum_{d} \pi(d, k \mid q) \big[P(C=1 \mid q, d, k) - \sigma(\theta_k + \gamma_{q,d}) \big] = 0. \tag{4.18}$$

For these gradients to vanish at the optimal parameters, the following condition must hold for every query-document pair with non-zero display probability, meaning when $\pi(d, k \mid q) > 0$:

$$P(C{=}1 \mid q, d, k) = \sigma(\theta_k + \gamma_{q,d}). \tag{4.19}$$

In a well-specified model, this condition can be satisfied for all query-document pairs where $\pi(d, k \mid q) > 0$. Thus, the influence of the logging policy on the estimated parameters vanishes as long as identifiability is guaranteed. □

Lemma 2 (Logging policy impact on misspecified models)**.** *When a two-tower model is misspecified, systematic correlations between the model's residual errors and the logging policy can introduce bias in parameter estimation, even when identifiability is guaranteed.*

Proof. When the model is unable to match the true click probabilities, we have a non-zero residual click prediction error:

$$\epsilon(q, d, k) \equiv P(C{=}1 \mid q, d, k) - \sigma(\theta_k + \gamma_{q,d}) \neq 0, \tag{4.20}$$

with the gradient conditions becoming:

$$\sum_k \pi(d, k \mid q)\, \epsilon(q, d, k) = 0, \quad \sum_q \sum_d \pi(d, k \mid q)\, \epsilon(q, d, k) = 0. \tag{4.21}$$

These conditions state that the policy-weighted averages of the residuals must vanish across positions and across items. However, when our model's click prediction errors $\epsilon(q, d, k)$ are systematically correlated with the policy's display probability $\pi(d, k \mid q)$, the optimizer must shift model parameters from their true values to satisfy these conditions. Note that model misspecification alone can bias model parameters as the average residual error might be non-zero even under a uniform random logging policy. But the logging policy can introduce additional bias in parameter estimation beyond what would be caused by model misspecification. □

Our analysis reveals a nuanced relationship between logging policies and additive two-tower models. While perfectly specified models remain unaffected by logging policies (beyond identifiability concerns), real-world models might inevitably contain some degree of misspecification. This creates a vulnerability where systematic correlations between model errors and logging policy behaviors can significantly distort parameter estimation. *The more a logging policy systematically favors certain documents for certain positions, the more these correlations can amplify estimation bias.*

4.5.3 Common sources of model misspecification

In the following, we highlight three scenarios where model misspecification creates errors systematically correlated with logging policies, leading to biased parameter estimation. We will omit a detailed mathematical analysis of each use case for brevity.

Functional form mismatch. Model misspecification arises when our model does not match the true data-generating process. For example, we fit a linear relevance model to data following non-linear patterns. If this mismatch leads to incorrect click predictions for query-document pairs with certain features and our logging policy associates these features with specific positions, we have a problematic correlation between our residuals and our logging policy, leading to biased model parameters.

Omitted variable bias. A second common problem is a logging policy having access to more information than our current model. If our logging policy used features that are predictive of user preference to place certain query-document pairs in certain positions (e.g., through business rules), but our current model does not have access to these features, our residual prediction errors will be systematically correlated with position due to omitted variable bias [190].

Expert policy in simulation. The third case is a subtle version of omitted variable bias, which is relevant in simulation experiments. In fact, it is the problem in our motivating example in Figure 4.1.[7] A simple method to simulate a strong logging policy used by related work is sorting by the expert annotated labels provided in classic learning-to-rank (LTR) datasets, i.e., sorting documents from most to least relevant [47, 200]. However, note that the features provided in real-world LTR datasets [24, 42, 145] cannot perfectly predict the expert relevance labels. Thus, sorting directly by these labels instead of sorting by a model trained on these datasets is essentially like a logging policy having access to a very predictive relevance feature that is being withheld from our current model, leading to omitted variable bias. In the following, we propose a simple method to reduce the effect of logging policies on misspecified two-tower models.

4.6 Sample Weights for Misspecified Models

While the focus of this chapter is analytical, our findings on logging policy effects suggest a fix for training two-tower models, leading us to **RQ3.3**. The priority should be to resolve model misspecification, as this lowers the impact of unequal exposure through the logging policy. However, should that not be possible (e.g., because business rules that impacted our past rankings were not documented), we can reduce the impact of uneven document distributions by weighting each query-document pair inversely to its propensity of being displayed:

$$\hat{\mathcal{L}}_{\text{IPS}}(\theta, \gamma : D) = -\frac{1}{N} \sum_{(q,d,k,c)\in\mathcal{D}} \frac{1}{\pi(d, k \mid q)} \big[c \log \sigma(\theta_k + \gamma_{q,d}) + (1-c)\log(1-\sigma(\theta_k+\gamma_{q,d}))\big]. \tag{4.22}$$

Note how this approach differs from traditional inverse propensity scoring (IPS) methods used in unbiased learning-to-rank [99, 137] as we target different biases: existing pointwise IPS methods primarily correct for position bias [13, 79, 157]. In contrast,

[7] We include results of our motivating example without an expert policy in Appendix 4.B.

this sample weight corrects for the uneven document distribution across ranks. The loss is essentially fitting a model against a version of the dataset in which all documents appeared equally across positions, which is related to corrections used in the position bias estimation literature [3, 14, 60], the policy-aware IPS estimator for selection bias [115, 137], or the recent work by Luo et al. [122]. Note that estimating this propensity is a challenge on its own. In this work, we naively calculate the propensities by counting how many times a query-document pair was displayed in a given position.

4.7 Experimental Setup

To evaluate our theoretical claims about model identifiability and logging policy effects, we implemented a comprehensive simulation framework. This controlled setting allows us to measure how well two-tower models can recover their underlying parameters under various conditions. Rather than solely relying on ranking metrics (such as nDCG), which can obscure underlying parameter estimation issues [47, 48], we directly compare inferred bias parameters against their ground-truth values to assess model identifiability. This provides clearer insights, as correctly estimated position bias parameters indicate properly identified relevance parameters due to their complementary relationship. By controlling logging policy strength, position randomization levels, and model specification, we will test each component of our theoretical analysis in isolation.

Datasets. The basis for our simulations are traditional learning-to-rank datasets with query-document features with an expert-annotated relevance label between 0-4. While we considered multiple datasets [24, 41, 42], we will focus the discussion in the rest of the chapter solely on MSLR30K [145] from the Bing search engine (31,531 queries and 136 dimensional feature vectors). We make this choice as our experiments consider very idealized simulations in which the choice of base dataset barely matters. We use the official training, validation, and testing splits. To increase computational efficiency, we truncate the number of documents per query to the 25 most relevant documents and drop all queries without a single relevant document ($\approx 3.1\%$ of queries). These steps purely aid the scalability of our simulation and all of our findings hold up on the full dataset. Next, we normalize each query-document feature using $\log 1\mathrm{p}(x) = \ln(1+|x|)\odot\mathrm{sign}(x)$ following Qin et al. [146].

Synthetic relevance. LTR datasets come with relevance labels obtained by human experts [24, 145]. To isolate the effect of model mismatch, we additionally generate synthetic relevance labels that follow a known model class based on the provided query-document features. We generate a linear relevance label with Gaussian noise:

$$\hat{\gamma}_{q,d} = w^T x_{q,d} + \xi_{q,d}, \quad \xi_{q,d} \sim \mathcal{N}(0, \sigma^2), \tag{4.23}$$

and a non-linear relevance label using a two-layer neural network with 16 neurons and tanh activations:

$$\hat{\gamma}_{q,d} = W_2^T \tanh(W_1^T x_{q,d} + b_1) + b_2 + \xi_{q,d}, \quad \xi_{q,d} \sim \mathcal{N}(0, \sigma^2). \tag{4.24}$$

In all our experiments we initialize the model weights randomly and add Gaussian noise with standard deviation $\sigma = 0.2$ to each label. Finally, we apply min-max scaling to ensure the resulting labels are in a comparable range to expert annotations, mapping the 5th to 95th percentiles of the synthetic labels to the range [0,4].

Logging policy. We isolate ranking performance and positional variability in our logging policy. To create a strong pointwise ranker, we train a relevance tower on the complete ground-truth relevance annotations of the training set with a mean-squared error loss. Note that we intentionally train our logging policy on the full training dataset and do not adhere to the common ULTR practice of training a weak ranker on 1% of the dataset [5, 93, 99, 137] as we need the strongest policy that our pointwise two-tower models could still capture. Our final logging policy score $s_{q,d}$ interpolates between our logging policy predictions and random noise:

$$s_{q,d} = \text{sign}(\alpha) \left(|\alpha| \cdot \hat{\gamma}_{q,d} + (1 - |\alpha|) \cdot u_{q,d} \right), \; u_{q,d} \sim U(0, 4), \tag{4.25}$$

where $\alpha \in [-1, 1]$ is a hyperparameter to set the logging policy *strength*. We simulate three levels of policy strength, using $\alpha = 1$ for the best possible ranking, $\alpha = 0$ for a random ranking, and $\alpha = -1$ for an inverse ranking that is worse than random by ranking documents from least to most relevant. Note that we sample noise only once per query-document pair and otherwise keep the ranking fixed across all user sessions for the same query. The noise interpolation setup follows Zhang et al. [200] but does not introduce the same problems as we sort by a trained logging policy and not directly by ground-truth expert labels (see Section 4.5.3).

Next, we transform our deterministic logging policy into a stochastic policy to introduce varying levels of positional variability into our simulation process, a crucial component to create overlap for our identifiability experiments. While various approaches exist to create stochastic policies, including the Gumbel-Max trick [19, 124] or randomization schemes from the position bias literature [99, 148], we opt for an epsilon greedy strategy for simplicity and interpretability [186]. With this approach, we show a uniform random ranking with probability τ, and otherwise rank deterministically according to our logging policy scores $s_{q,d}$. We refer to this probability τ as the *temperature* of our logging policy, with $\tau = 0$ indicating a deterministic policy (showing the same ranking across all sessions) and $\tau = 1$ indicating a uniform random ranking per session.

Click simulation. We simulate user clicks with position bias following the two-tower model and leave an investigation of user model mismatch to future work. We define position bias logits as: $\hat{\theta}_k = -\ln(k)$ where k is the rank of the current query-document pair. The final click probability is: $P(C = 1 \mid q, d, k) = \sigma(\hat{\theta}_k + (\hat{\gamma}_{q,d} - 2))$, with our final relevance logits being a zero-centered version of our relevance labels obtained earlier. We center the scores from their original range between [0, 4] to [-2, 2] to avoid click logits that might overly saturate the sigmoid. For all experiments, we simulate 1M user sessions for training and 500K sessions for validation and testing respectively. We repeat all simulations over three random seeds and plot the 95% confidence interval across all figures.

Model. Lastly, we describe our model architecture. Our bias tower uses a single parameter per rank as the position bias logit θ_k. Our relevance tower is either a single embedding parameter for each query-document pair $\gamma_{q,d}$, a linear model, or a two-layer feed-forward neural network (2 layers, 32 neurons, ELU activations). All models were optimized using AdamW [121] with a learning rate of 0.003 and a weight decay of 0.01 up to 50 epochs, stopping early after three epochs of no improvement of the validation loss. The entire setup was implemented in Jax [18], Flax NNX [86], and Rax [94].

4.8 Results

In the following, we empirically test the claims of our work. First, we will examine our theory of identification and the claim that logging policies do not impact well-specified two-tower models, meaning when our model assumptions match the actual user behavior. Second, we implement the three settings described in Section 4.5.3 to investigate if logging policies amplify biases under model mismatch. Lastly, we evaluate whether the sample re-weighting scheme described in Section 4.6 can alleviate the logging policy impact in each of the three model mismatch settings. Due to spacial constrains, we will only visualize models trained on the strongest logging policies (with varying temperature parameters), as this is the setting for which we expect the largest logging policy effects. Full evaluation results are available in our online repository.

4.8.1 Evaluating well-specified models

First, we evaluate our theory of identifiability stating that two-tower models with separate relevance parameters per query-document pair require exact document swaps. We display position bias estimated by a two-tower model trained with separate query-document parameters in the right panel of Figure 4.2. The model fails to infer the simulated bias under a deterministic policy ($\tau = 0$) when no document swaps are available. All other temperature settings introducing swaps across positions allow model identification.

The second part of our identifiability theory states that two-tower models can be identified from query-document features, as long as there are overlapping feature distributions between ranks. The left panel in Figure 4.2 displays a linear relevance tower trained on clicks following a synthetic linear relevance model, and the middle panel a model using a neural relevance tower trained on users following a non-linear relevance distribution (both generated as described in Section 4.7). The results confirm that under idealized conditions, model parameters can be perfectly recovered even on a fully deterministic policy, showcasing that features can fully identify two-tower models.

The last observation we can make in Figure 4.2 is related to our claim that logging policies have no effect on well-specified two-tower models that capture the true data generating process. Once identifiability is guaranteed (through feature overlap or explicit document swaps), all three models are completely unaffected by different levels of simulated randomization that coincide with different logging policy performance. Our first experiment confirms our theory on identification as well as our claim that logging policies do not affect well-specified two-tower models once identifiability is guaranteed.

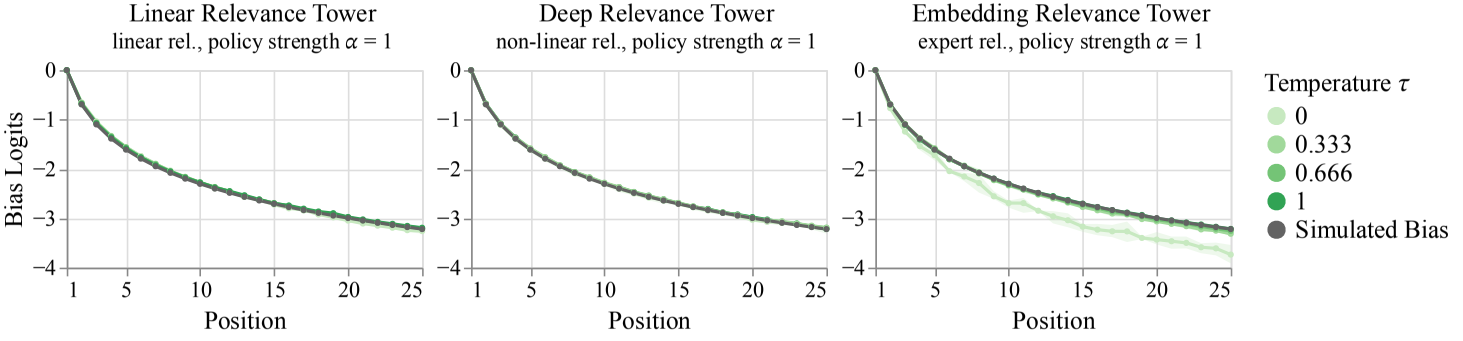


Figure 4.2: Evaluating position bias for three well-specified models matching the simulated user behavior. All models converge to ground-truth bias parameters while being trained on the strongest logging policy ($\alpha = 1$). Feature-based models recover position bias without any document swaps ($\tau = 0$), while the embedding-based model requires swaps for identification ($\tau > 0$).

4.8.2 Evaluating misspecified models

Next, we investigate three scenarios of systematic model misspecification which, based on our theory in Section 4.5.2, should introduce a bias that should be amplified by our logging policy. Figure 4.3 displays the resulting position bias parameters. In the left panel, we simulate fitting an incorrectly specified relevance model by training a linear relevance tower on data following a non-linear relevance behavior. In the middle panel, we simulate omitted variable bias by training a logging policy on all 136 query-document features of MSLR30K but make only the first 100 features available to the downstream two-tower model. And the right panel creates model misfit by using the simulation setup of Deffayet et al. [47] and Zhang et al. [200], who rank query-document pairs based on expert-annotated relevance labels to create a strong logging policy. In all three cases, we first observe a notable bias in the estimated parameters that coincides with logging policy correlation (a higher temperature implies more randomness and less correlation). Secondly, we observe that a lower temperature (which creates a more unequal query-document distribution across ranks) amplifies bias. In other words, as rankings deviate more from uniformly random exposure, the magnitude of bias increases. Importantly, this bias manifests itself just as strongly when simulating an inverse logging policy that deliberately positions all relevant documents at the bottom (Appendix 4.C). This confirms that non-uniform query-document exposure across ranks can intensify biases in misspecified two-tower models. Lastly, we can also see that a uniform random ranking ($\tau = 1$) gets close but does not perfectly match the simulated bias, even under these highly idealized conditions. This confirms that model misfit on its own introduces bias, independent of logging policy effects.

4.8.3 Addressing unequal exposure using IPS

Our last experiment tests if the sample weighting scheme described in Section 4.6 can help alleviate the effects of unequal document exposure on misspecified models. Figure 4.4 displays the results of applying the weighting scheme to the same three simulation setups used in our previous model mismatch experiments. We see that the weighting scheme does not work under a deterministic logging policy $\tau = 0$, as documents have a probability of one to be displayed in their initial rank and a propensity of zero to be displayed in any other rank. Propensity scoring requires positivity [40], meaning in our case that documents have a non-zero chance to be displayed at other ranks. We see that in all other cases when we introduce increasing levels of variability into our policy, the models converge to identical position bias estimates, independent of the logging policy. Note that the estimated parameters are still slightly biased due to model misspecification, however, the bias amplification by the logging policy has been dampened. While this setting is highly idealized, it confirms that weighting query-document pairs inversely to their propensity of being displayed at a given rank, can help counterbalance unequal exposure effects in two-tower models.

4.9 Discussion

In the following, we discuss the practical implications and the limitations of this chapter.

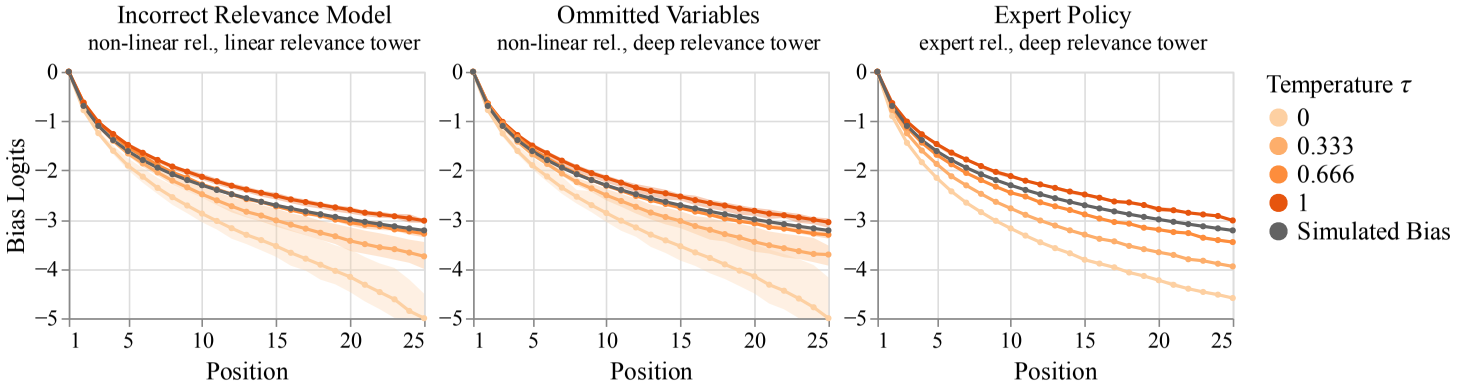


Figure 4.3: Evaluating position bias under model misspecification: (left) training a linear relevance model on non-linear user behavior, (middle) using fewer features when training two-tower models than are available to the logging policy, and (right) simulating an oracle logging policy by sorting by expert relevance labels from MSLR30K. The logging policy amplifies bias in all models. Model misspecification is visible as estimations do not match the simulated bias on fully randomized data ($\tau = 1$).

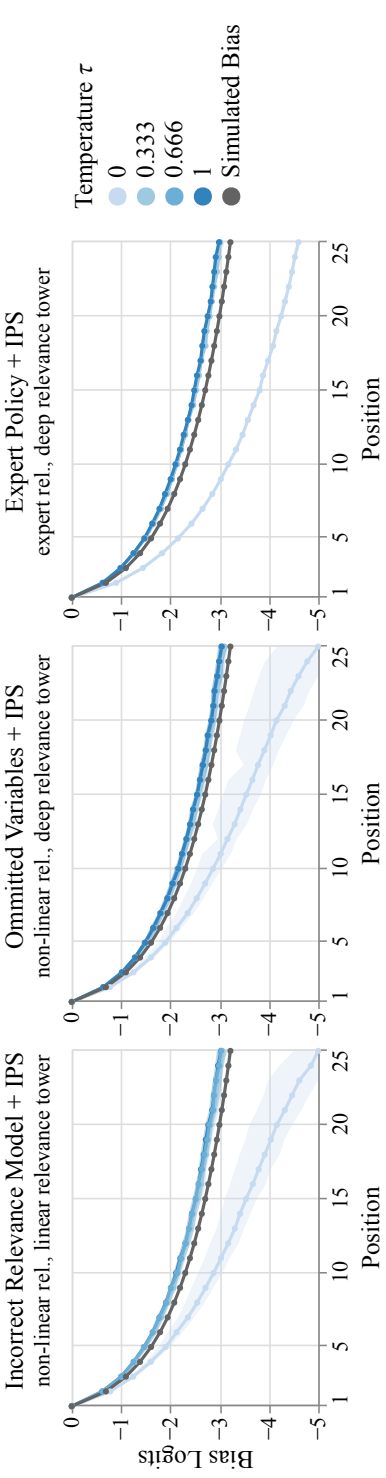


Figure 4.4: Reweighting samples during training inversely to the propensity of a policy placing an item in a given position. IPS requires that documents have a non-zero chance of occurring in other rank. Therefore, we see a helpful impact of IPS across various levels of randomization, but no improvement on a deterministic policy ($\tau = 0$). Note that while the logging policy impact is minimized, a bias remains in all three examples due to model misspecification.

4.9.1 Practical implications

Our theory offers several important implications for those working with two-tower models. First, we find that models generalizing over query-document features require overlapping feature distributions across ranks to ensure identifiability. Second, while logging policies do not impact well-specified two-tower models, they can significantly amplify bias in models that systematically fail to capture the actual user behavior. Therefore, we advise determining the underlying user behavior model (e.g., using randomized experiments) and continuously monitoring click prediction residuals for correlations with the logging policy to detect emerging mismatches between model assumptions and reality. Accurately modeling users might create tensions: practitioners may be tempted to condition on more features to improve click predictions and minimize model misfit, but increasing feature dimensionality reduces overlap between ranks, potentially compromising model identifiability. Similarly, reducing the expressivity of model towers (e.g., using shallow networks and regularization) can improve identifiability but risks underfitting the data, leading to misspecification bias that logging policies may amplify.

Collecting randomized data helps address these issues by creating overlap and guaranteeing identifiability while reducing unequal document exposure across ranks, thus lessening the impact of model misspecification. However, this must be balanced against a potential disruption of the user experience. For researchers working in simulation, we caution against creating logging policies by directly sorting documents according to expert annotations. This approach introduces a subtle form of omitted variable bias, simulating a logging policy with access to perfect relevance information that the two-tower model cannot capture. Instead, researchers should train a separate logging policy model from features. Finally, when some degree of model misspecification is unavoidable, consider propensity weighting to counteract potential bias amplification through the logging policy.

4.9.2 Limitations

This chapter has several limitations that motivate directions for future research. First, we focus on identifiability, a concept fundamentally evaluated in idealized conditions of a well-specified model and infinite data [112]. While we confirmed our theories empirically, our simulations represent highly idealized settings tied specifically to additive two-tower models. Future work might develop empirical tests for the identifiability of models on specific real-world datasets (practical identifiability [152]). Note that even when data collection and model assumptions match in theory, empirical datasets might be too small (e.g., containing too few swaps across positions) to practically identify a two-tower model. Second, we focused on a single prevalent user model. Future research could explore how various user models interact with logging policies or consider empirical tests for identifiability that allow plugging in other models with different assumptions. Third, while we show that similar query-document features across ranks can identify two-tower models, future work might develop diagnostic metrics to quantify the overlap between high-dimensional query-document feature distributions on a given dataset. Promising directions to measure overlap include distance measures [68, 110], hypothesis testing [68, 120], or positivity measures from causal

inference [40]. Lastly, our propensity scoring method mainly demonstrates that we identified the correct mechanism. To be practically useful, future research might consider better propensity estimation techniques and variance reduction methods.

4.10 Conclusion

This chapter aims to answer **RQ3** on when two-tower models can reliably disentangle bias and relevance signals from clicks, and the impact of logging policies that collect clicks on two-tower models. We answer this question by first analyzing under which conditions we can identify the parameters of additive two-tower models from observational data (**RQ3.1**). Extending prior work, we show that two-tower models can be identified from rankings without document swaps, provided that feature distributions across ranks overlap. However, this identification relies on two key assumptions: a continuous relevance model and sufficient overlap across bias dimensions. In practice, overlap between feature distributions decreases with increasing feature dimensionality, and deep neural networks can produce large jumps in predictions for similar feature inputs despite being continuous, suggesting that explicit randomization through document swaps remains a vital tool to ensure identifiability in many production settings.

Second, we study the impact of logging policies on parameter estimation in two-tower models (**RQ3.2**). Contrary to existing work, we find that logging policies do not affect two-tower models beyond identifiability concerns, provided our model assumptions match actual user behavior, i.e., the model is well-specified. Our analysis indicates that existing work observed a deteriorating ranking performance of two-tower models [200], primarily due to a faulty simulation setup rather than an inherent confounding problem caused by strong logging policies.

However, we also show that misspecified two-tower models that make systematic click-prediction errors lead to biased parameter estimates. If click prediction errors correlate with document placement across positions, we find that logging policies can amplify this problem and lead to stronger bias in model parameters, as demonstrated by three examples. Meaning, we do find an effect of logging policies on two-tower models, but only under a specific kind of model misspecification.

Lastly, we show that weighting each query-document pair by its probability of being displayed to the user can help reduce additional bias introduced by the logging policy on misspecified two-tower models (**RQ3.3**). However, the method can only reduce and not fully mitigate biases introduced by model misspecification.

This chapter focused solely on understanding additive two-tower models, raising the broader question of whether we can apply the principles behind two-tower models, neural parameterization and gradient-based optimization, to a broader class of click models that capture richer user behavior. The next chapter addresses this question and introduces a framework for neural click modeling beyond two-tower models at scale.

4.11 Reproducibility

All code, data, and complete simulation results for this chapter are available at:
https://github.com/philipphager/two-tower-confounding

4.A Bias Features beyond Position

Two-tower models owe their popularity in industry applications partially to their ability to use bias features beyond position that might impact an item's examination probability [33, 195]. These features might include, a.o., device platform (mobile vs. desktop), content type (text vs. video), or the display height of the document on screen [209]. Such information can be captured in a bias feature vector $z_{q,d} \in \mathbb{R}^n$ serving as input to a dedicated bias tower $b(\cdot)$, commonly implemented a second neural network:

$$P(C = 1 \mid q, d) = \sigma(b(z_{q,d}) + r(x_{q,d})). \tag{4.26}$$

In this expanded model, all discussed identifiability principles from Section 4.4.4 still apply but with increased combinatorial complexity. The model remains identifiable only when sufficient overlap exists in document distributions across the entire space of bias features:

$$\text{supp}\left(P(x \mid z)\right) \cap \text{supp}\left(P(x \mid z')\right) \neq \emptyset. \tag{4.27}$$

This requirement presents a combinatorial challenge, as the number of potential bias configurations grows exponentially with each additional bias dimension.

To illustrate, consider a setting with five positions, three content types, and two device types. This creates 30 distinct bias configurations that must form a connected graph through overlap in the query-document feature distribution. Without careful data collection, certain bias combinations may remain isolated, leading to unidentifiable parameters in those regions of the feature space. In practice, this may require explicit randomization across bias dimensions or the collection of vast datasets to capture enough natural variation across all bias configurations.

4.B Motivating Example without Expert Policy

We revisit our motivating example from the introduction and replace the expert policy (sorting by ground-truth annotations) with a deep neural network trained on expert annotations. We can see in Figure 4.B.1 that removing this problematic source of model misspecification from our simulation greatly reduces the observed logging policy on ranking performance. We conclude that the original observation from Zhang et al. [200] replicated earlier in Figure 4.1 is mostly a relic of an improper simulation setup.

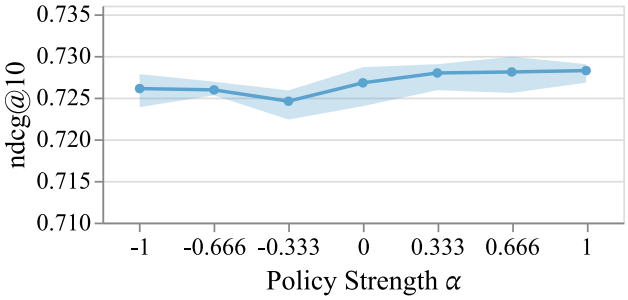


Figure 4.B.1: Two-tower models trained on deterministic logging policies of varying strengths α. The logging policy is a deep relevance tower trained on expert annotations instead of directly sorting by ground-truth relevance labels, as done in the introduction.

4.C Extended Model Misspecification Simulations

In the following, we provide more visualizations of our three cases of model misspecification. First, we display position bias logits estimated on the inverse policy in Figure 4.C.1, i.e., when placing the least relevant items on top which is worse than random ranking and creates correlation between document placement and the model's prediction errors. Second, we display the corresponding ranking performance in terms of nDCG@10, measured against the respective relevance label used in each experiment in Figure 4.C.2.

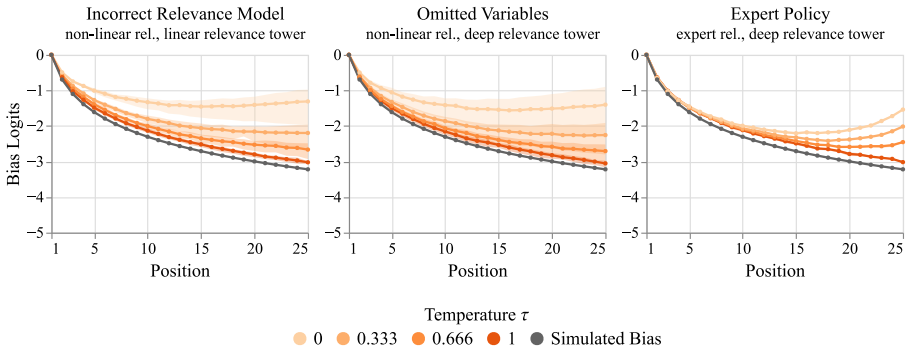


Figure 4.C.1: Position bias estimation under model misspecification on an inverse policy ($\alpha = -1$) placing the most relevant items at the bottom: (left) training a linear relevance model on non-linear user behavior, (middle) training the logging policy on more features than available to the two-tower model, and (right) sorting by expert relevance labels from MSLR30K as logging policy. The results confirm recent work on propensity over/underestimation by Luo et al. [122].

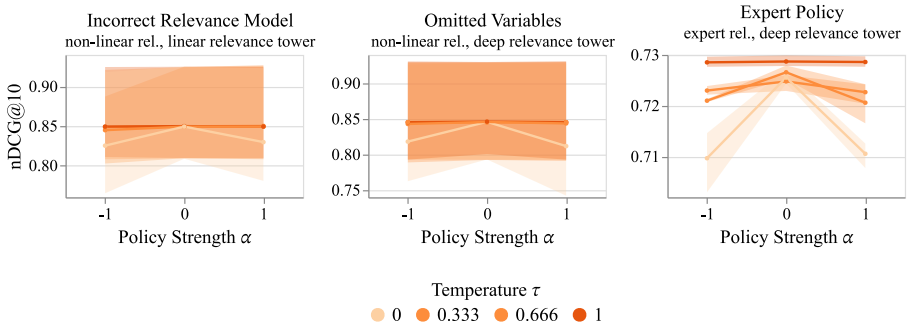


Figure 4.C.2: Ranking performance of the relevance tower under model misspecification across different logging policy strengths and temperatures. While the overall ranking performance is significantly lower than the ideal (an nDCG@10 of 1 is possible in the left and middle panel), ranking performance does not always correspond to the clear bias in position parameters highlighted in Figure 4.3 and Figure 4.C.1, at least not on our dataset size. This echos recent findings on the unreliability of evaluating click model parameters through nDCG [47, 48].

5

CLAX: Fast and Flexible Neural Click Models in JAX

The two-tower models we have encountered in previous chapters are popular bias correction methods due to (i) their flexibility to include custom neural architectures and features, and (ii) their scalability through GPU-accelerated gradient-based training enabled through modern deep learning frameworks. However, two-tower models follow the position-based model, which makes strong assumptions on user behavior, most notably that users examine and click on documents independently [153]. In reality, users might compare documents or stop examining other documents after clicking [39, 56, 70]. Over time, many click models more complex than the PBM have been introduced: The dynamic Bayesian network (DBN) [25] models user satisfaction after clicking a document, the user browsing model (UBM) [56] considers examination dependencies across positions, and the click chain model (CCM) [70] models users abandoning a session. These models are traditionally formulated as probabilistic graphical models (PGMs) [109] with latent variables and optimized using expectation maximization (EM) [37]. However, the iterative nature of EM scales poorly with dataset size without specialized algorithms [102, 127, 171] and requires model-specific derivations [37].

This chapter investigates whether we can systematically apply the ideas behind two-tower models, neural parameterization and gradient-based optimization, to a broader class of PGM-based click models, potentially accounting for more complex biases in user behavior while maintaining computational efficiency. This leads us to ask:

RQ4 Can gradient-based optimization replace Expectation-Maximization to enable scalable and flexible neural parameterization of PGM-based click models?

We make three contributions to answer this question. First, we theoretically compare EM-based and gradient-based optimization for PGM-based click models. Second, we introduce CLAX, a framework for training PGM-based neural click models with automatic differentiation in JAX. Using CLAX, we implement neural variants of ten common click models and evaluate their click prediction accuracy and scalability against a well-established EM baseline. Third, we investigate whether more expressive neural click models, beyond the PBM, improve performance in a learning-to-rank setting.

This chapter is currently under review and available as: P. Hager, O. Zoeter, and M. de Rijke. CLAX: Fast and flexible neural click models in JAX. *arXiv preprint arXiv:2511.03620*, 2025.

Listing 5.1: A minimal example of training a UBM in CLAX.

```
from clax import Trainer, UserBrowsingModel
from flax import nnx
from optax import adamw

model = UserBrowsingModel(
    query_doc_pairs=100_000_000,
    positions=10,
    rngs=nnx.Rngs(42),
)
trainer = Trainer(
    optimizer=adamw(0.003),
    epochs=50,
)
train_df = trainer.train(model, train_loader, val_loader)
test_df = trainer.test(model, test_loader)
```

5.1 Introduction

Click models are an integral part of web search and recommender systems, serving as fundamental tools for understanding and predicting user interactions [37]. Click models are used to determine user behavior on search engine result pages [25, 39, 56, 153], predict clicks in advertising [34, 129, 205], estimate click biases for counterfactual learning-to-rank methods [176, 184], simulate users in reinforcement learning [49, 91], evaluate new ranking systems [36], and are directly used as ranking models [195].

Early click models can be represented as probabilistic graphical models (PGMs) [37] that explicitly model variables such as document attractiveness [39, 153], user satisfaction [25, 70], and rank examination [56, 153] that might influence user behavior. Since most of these variables are latent, popular models including the position-based model (PBM) [153], user browsing model (UBM) [56], and dynamic Bayesian network (DBN) of [25] are optimized using expectation maximization (EM). EM iteratively optimizes probabilistic models with missing variables by alternately imputing missing values and updating parameters, and is widely employed in click modeling libraries like PyClick[1] or ParClick [102]. The iterative nature of EM, however, scales poorly with dataset size, motivating specialized algorithms [102, 171] and online optimization procedures [127].

Over the last decade, neural click models have emerged along two directions. The first uses neural architectures such as recurrent neural networks [17, 28], attention [206], or graph neural networks [117] to improve click prediction and ranking performance, potentially sacrificing the interpretability that made traditional PGMs valuable for understanding user behavior. The second branch parameterizes simple PGMs with deep neural networks [48, 195]. A prominent example is the two-tower model, a neural parameterization of the PBM [153], popular for position bias correction in industry [72, 85, 195, 202].

However, a significant gap remains: *the paradigm shift to gradient-based optimization has not been systematically applied to more complex PGM-based click models [37], preventing practitioners from leveraging modern deep learning frameworks while pre-*

[1] https://github.com/markovi/PyClick

serving the interpretability and theoretical foundations of classic models. Given that simple two-tower models already demonstrate strong empirical ranking and click prediction performance [72, 80, 85, 195], parameterizing more sophisticated models like the DBN [25] and UBM [56] may yield significant performance improvements.

Modern frameworks such as JAX [18] are compelling for addressing this gap. JAX's functional programming and automatic differentiation eliminate the need for manual gradient derivations in complex probabilistic models, while just-in-time (JIT) compilation and vectorization enable efficient computation on large datasets. Building on the JAX-based deep learning framework Flax NNX [86], we introduce CLAX: A neural click modeling library that bridges traditional PGM-based click models with modern gradient-based optimization, replacing EM with direct optimization of marginal log-likelihoods in a numerically stable way.

CLAX is designed around modularity and scale. By default, CLAX uses embeddings to represent model components such as the attractiveness of individual query-document pairs, making it equivalent to classic click modeling libraries [37]. Listing 5.1 shows a minimal example of training a UBM with CLAX. However, CLAX's modular design allows broad customization beyond embeddings by enabling the use of linear models, deep networks, deep cross networks [182], or custom Flax modules. Any module matching the output shapes expected by CLAX can be plugged in for end-to-end optimization. This modularity even allows meta-models, which we demonstrate by building on ideas by Yan et al. [195] and training a mixture model over multiple click models for datasets where users exhibit different behaviors across sessions.

To achieve scale, CLAX leverages baseline correction and embedding compression techniques from the deep recommender systems literature [163, 188] combined with JAX's computational efficiency for training on billions of query-document pairs on a single GPU. We demonstrate this scalability by training and evaluating on the complete Baidu-ULTR dataset [209] containing over 1 billion user sessions in under 2 hours.

Beyond scale, we evaluate CLAX models in a learning-to-rank setting and find that neural parameterizations of sophisticated PGMs exhibit higher ranking performance than widely-used two-tower models, demonstrating that increased model complexity can improve ranking performance in real-world settings.

Our contributions are fourfold:

- We demonstrate that direct gradient-based optimization can replace EM for training traditional PGM-based click models and achieve comparable empirical click prediction performance.
- We introduce CLAX, a JAX-based click modeling library with a highly modular design allowing any component to be plugged into classic PGM structures.
- We showcase the computational efficiency of CLAX by training and evaluating on over 1 billion user sessions of the Baidu-ULTR dataset on a single GPU.
- We show that neural parameterizations of sophisticated PGMs can surpass the ranking performance of widely-used two-tower models, potentially providing a powerful new tool for practitioners.

CLAX is meant as a tool for practitioners and researchers alike. Industry practitioners might extend two-tower models and use cascade-like models. Researchers benefit from the demonstration of how the marginal log-likelihood of many PGMs can be optimized directly using SGD and autograd frameworks, simplifying the creation of new models. The core paradigm of CLAX is not tied to JAX. All models can be implemented in PyTorch or TensorFlow. We chose JAX for its speed and growing ecosystem [46].

Below, we first cover related work on click modeling, implementations in the field, and the JAX ecosystem. Section 5.3 compares gradient-based with EM-based optimization for click models. Section 5.4 provides an overview of the CLAX library and Section 5.5 discusses the basics of its numerically stable implementation. Lastly, we evaluate CLAX empirically in Sections 5.6 and 5.7.

5.2 Related Work

5.2.1 Click models

Click models emerged as probabilistic graphical models (PGMs) to predict user clicks on search engine result pages [37]. Most click models are extensions of two models. The position-based model (PBM) [153] assumes that users click after examining the position of a document and finding the document attractive. Notably, observing and clicking on a document under the PBM is independent of all other documents in the same ranking. The cascade model [39] is the second foundational model, assuming that users examine items sequentially from top to bottom, click on the first relevant item, and then stop browsing. Note that the cascade model can only explain a single click per result page.

The dependent click model (DCM) [71] extends the cascade model to account for multiple clicks by assuming a rank-dependent continuation probability, whereby users may continue browsing after clicking on a document. The click chain model (CCM) [70] extends this idea by assuming users continue after a click based on the attractiveness of the current document. The dynamic Bayesian network (DBN) [25] further extends this idea by separating document attraction (before click) and satisfaction (which is revealed after clicking). A popular extension, the SDBN assumes that users always continue browsing when they are not satisfied with the current item, whereas the DBN learns a separate continuation parameter. The user browsing model (UBM) [56] extends the PBM by assuming that examining a document not only depends on the current position but also on the position of the last clicked document, thereby relaxing the independence assumption of the PBM.

CLAX implements seven foundational PGM-based click models and three CTR-based baselines, as covered by Chuklin et al. [37], using gradient-based optimization. We provide the complete derivation of each model and its marginal log-likelihood in Appendix 5.A.

5.2.2 Neural click models

In recent years, click models have incorporated neural networks in two ways. First, new click models based on neural architectures have emerged. Models like the NCM

[17] or the CACM [28] predict clicks using recurrent neural networks. The XPA model by Zhuang et al. [206] uses attention to capture interactions between documents in a grid layout. And the GraphCM [117] leverages graph neural networks to model user behavior across sessions. These methods demonstrate strong empirical performance in click prediction at the risk of being less interpretable than PGM-based click models. While these models can be implemented with Flax NNX and CLAX, our focus in this work is on gradient-based optimization of classic PGM-based models.

Second, neural implementations of classic click models have emerged. Neural networks based on the PBM are known as two-tower models and have become prevalent in industry ranking settings [72, 85, 195], allowing the use of document features and custom network architectures to predict examination and attractiveness. CLAX provides the logical next step by enabling easy parameterization of more advanced click models.

Third, we acknowledge that we are not the first to apply gradient-based optimization to PGM-based click models beyond the PBM. Deffayet et al. [48] implement the PBM, UBM, and DBN using gradient-based optimization in PyTorch while investigating click model robustness to distributional shift. However, their work focused on robustness analysis rather than validating against EM baselines or providing a general-purpose library. CLAX extends this work by (i) empirically demonstrating equivalent click prediction performance to EM baselines, (ii) providing a comprehensive library with an API that enables flexible neural parameterization, and (iii) demonstrating scalability to billion-scale datasets.

5.2.3 Frameworks for click modeling

The standard library for click modeling is PyClick,[2] published by Chuklin et al. [37], which optimizes click models using maximum likelihood estimation and EM. While their iterative EM optimization can be considerably accelerated with the PyPy[3] interpreter, it remains too slow even for medium-scale datasets (see Section 5.7.1). To address these performance limitations, specialized libraries have emerged. ParClick [102][4] is a C++ library that parallelizes EM across multiple CPU cores on a single machine. MassiveClicks [171][5] extends ParClick to support multi-node and GPU-based training, achieving substantial improvements in training time. However, these specialized libraries focus primarily on EM-style optimization, making them difficult to extend with custom neural modules or use additional features.

Rather than competing with specialized libraries on computational speed, CLAX takes a different approach. We implement all PyClick models, replacing EM with gradient-based optimization using a general-purpose deep learning framework. This enables the end-to-end optimization of fully customizable models, while benefiting from JAX's speed through just-in-time compilation and GPU support [18, 86]. Our JAX implementation demonstrates a paradigm that can be translated to other deep learning frameworks, such as PyTorch or TensorFlow.

[2] https://github.com/markovi/PyClick
[3] https://www.pypy.org/
[4] https://github.com/uva-sne/ParClick
[5] https://github.com/skip-th/MassiveClicks

5.2.4 The JAX ecosystem

JAX is a Python library for numerical computation built around composable function transformations that enable automatic differentiation, JIT-compilation to GPU/TPUs, and support for vectorized and distributed computing [18]. Unlike monolithic frameworks such as PyTorch or TensorFlow, JAX adopts a modular ecosystem approach where functionality is distributed across specialized libraries [46]. The base JAX library provides a NumPy-compatible interface and fundamental transformations, while additional libraries offer domain-specific functionality: Flax NNX [86] and Haiku [87] are neural network libraries, Optax supplies optimizers, Chex offers testing utilities, and Distrax provides probability distributions [46]. This modular design enables research communities to develop specialized tools for their respective field [22, 106, 147]. In information retrieval, Rax [94] is the primary library for learning-to-rank loss functions and evaluation metrics. CLAX is the first click modeling framework in the ecosystem, integrating with Optax for optimization, Flax NNX for deep learning, and Rax for ranking metrics.

5.3 From EM to Gradient-based Optimization

Next, we compare the expectation maximization algorithm [50] and gradient ascent[6] for inferring parameters in click models, using the example of the position-based model (PBM) [39, 153]. We focus our analysis on a single query with documents $d \in D$ displayed at positions $k \in \{1, \ldots, K\}$ to simplify our notation. The PBM assumes that a user clicks on a document d if they examine its position k and find it attractive.[7] We define binary random variables for click C, examination E, and attractiveness A, with realizations c, e and a. The click probability of the PBM is:

$$P(C = 1 \mid d, k) = P(E = 1 \mid k) \cdot P(A = 1 \mid d) = \theta_k \gamma_d, \tag{5.1}$$

where θ_k is the probability of examining rank k and γ_d is the attractiveness probability of document d. Like many click models, the PBM contains latent variables. We only observe clicks, not whether a user examined a document or found it attractive. This means we cannot directly optimize the complete-data log-likelihood. Instead, we must find parameters that maximize the log-likelihood of the data we can actually observe, the marginal log-likelihood:

$$\mathcal{L}(\theta, \gamma; \mathcal{D}) = \sum_{(d,k,c) \in \mathcal{D}} \left[c \log(\theta_k \gamma_d) + (1 - c) \log(1 - \theta_k \gamma_d) \right]. \tag{5.2}$$

5.3.1 Expectation-maximization (EM) algorithm

A prominent method for optimizing likelihoods with latent variables is the EM algorithm [50]. Starting from an initial guess for our model parameters, EM cycles between

[6] In this section, we frame the problem as likelihood maximization, although in practice we minimize the negative log-likelihood using gradient descent.

[7] More commonly in connection with the PBM is the term "relevant." But as the click models in this chapter differentiate between users being attracted to a document snippet and being satisfied with a document after clicking, we follow Chuklin et al. [37] and use the more precise terminology of "attractiveness."

two steps until convergence. In the expectation step, we use the current model parameters and observed data to compute the posterior expectations of our latent variables. This step fills in the missing observations in our dataset. In the maximization step, we use these expected values to find new parameters that maximize the log-likelihood of the complete data. We use the newly obtained parameters to refine our posteriors in the next E-step, which leads us to find better fitting parameters in the next M-step, and so on. In the following, we showcase this principle for the PBM.

E-step:

Given parameters of the PBM $(\theta^{(t)}, \gamma^{(t)})$ at iteration step t and our observed clicks, we compute the posterior expectation of each latent variable for each observation $(d, k, c) \in \mathcal{D}$. For the binary variables of examination E and attractiveness A, this is equivalent to their posterior probability given the observed click c:

$$\begin{aligned}
\hat{e}_{d,k,c} &= \mathbb{E}[E \mid c, d, k; \theta^{(t)}, \gamma^{(t)}] \\
&= c \cdot P(E = 1 \mid C = 1) + (1 - c) \cdot P(E = 1 \mid C = 0) \\
&= c \cdot 1 + (1 - c) \cdot \frac{P(C = 0 \mid E = 1, d, k) P(E = 1 \mid k)}{P(C = 0 \mid d, k)} \\
&= c + (1 - c) \cdot \frac{(1 - \gamma_d^{(t)}) \theta_k^{(t)}}{1 - \theta_k^{(t)} \gamma_d^{(t)}}, \qquad (5.3) \\
\hat{a}_{d,k,c} &= \mathbb{E}[A \mid c, d, k; \theta^{(t)}, \gamma^{(t)}] \\
&= c + (1 - c) \cdot \frac{(1 - \theta_k^{(t)}) \gamma_d^{(t)}}{1 - \theta_k^{(t)} \gamma_d^{(t)}}. \qquad (5.4)
\end{aligned}$$

M-step:

In the maximization step, we use the posterior expectations $(\hat{e}_{d,k,c}, \hat{a}_{d,k,c})$ computed for each observation in the E-step at time t to find new model parameters by maximizing the expected complete-data log-likelihood ($\mathcal{Q}$-function):

$$\begin{aligned}
&\mathcal{Q}\left(\theta^{(t+1)}, \gamma^{(t+1)} \mid \theta^{(t)}, \gamma^{(t)}\right) = \\
&\sum_{(d,k,c) \in \mathcal{D}} \left[\hat{e}_{d,k,c} \log(\theta_k^{(t+1)}) + (1 - \hat{e}_{d,k,c}) \log(1 - \theta_k^{(t+1)}) \right] + \\
&\sum_{(d,k,c) \in \mathcal{D}} \left[\hat{a}_{d,k,c} \log(\gamma_d^{(t+1)}) + (1 - \hat{a}_{d,k,c}) \log(1 - \gamma_d^{(t+1)}) \right].
\end{aligned} \qquad (5.5)$$

Note that the $\mathcal{Q}$-function is commonly much simpler than our marginal log-likelihood. In the case of the PBM, it allows us to decouple the estimation of θ and γ. By taking the derivative of $\mathcal{Q}$ with respect to each parameter, setting it to zero, and solving, we

obtain closed-form update rules for the PBM:

$$\begin{aligned}\theta_k^{(t+1)} &= \frac{\sum_{(d,k',c)\in\mathcal{D},k'=k} \hat{e}_{d,k,c}}{\sum_{(d,k',c)\in\mathcal{D},k'=k} 1}, \\ \gamma_d^{(t+1)} &= \frac{\sum_{(d',k,c)\in\mathcal{D},d'=d} \hat{a}_{d,k,c}}{\sum_{(d',k,c)\in\mathcal{D},d'=d} 1},\end{aligned} \tag{5.6}$$

which divides the sum of all expected posterior examination values by the number of documents at a given position, and the sum of all expected attractiveness values for a document by the number of impressions of that document.

5.3.2 Gradient-based optimization

An alternative to EM is to optimize the marginal log-likelihood $\mathcal{L}$ in Eq. 5.2 directly using gradient-based methods. This involves computing the partial derivative of $\mathcal{L}$ with respect to each parameter and taking a step in the direction of the gradient [156]:

$$\frac{\partial \mathcal{L}}{\partial \gamma_d} = \sum_{(d',k,c)\in\mathcal{D},d'=d} \left(\frac{c}{\gamma_d} - \frac{(1-c)\theta_k}{1-\theta_k\gamma_d} \right), \tag{5.7}$$

$$\frac{\partial \mathcal{L}}{\partial \theta_k} = \sum_{(d,k',c)\in\mathcal{D},k'=k} \left(\frac{c}{\theta_k} - \frac{(1-c)\gamma_d}{1-\theta_k\gamma_d} \right). \tag{5.8}$$

We update model parameters iteratively using learning rate η:

$$\theta_k^{(t+1)} = \theta_k^{(t)} + \eta \frac{\partial \mathcal{L}}{\partial \theta_k} \quad \text{and} \quad \gamma_d^{(t+1)} = \gamma_d^{(t)} + \eta \frac{\partial \mathcal{L}}{\partial \gamma_d}. \tag{5.9}$$

5.3.3 Comparison and discussion

Both EM and gradient-based optimization are valid methods for finding maximum likelihood estimates when dealing with marginal log-likelihoods [109, Chapter 19]. While both can become trapped in local optima, they differ in their optimization characteristics. EM guarantees monotonic improvement in the marginal log-likelihood at each iteration, and it circumvents the complexity of directly optimizing the marginal likelihood by optimizing a simpler auxiliary function [50, 158, 191]. However, EM can be slow to converge in settings with high missing information [50, 191], and classical implementations require full dataset passes, a limitation that motivated online and stochastic variants [23, 131, 170].

Gradient-based methods offer different trade-offs, using general-purpose optimization advances (e.g., adaptive learning rates [55, 108, 121], momentum [144]) that can lead to fast convergence, without however, the guarantee of monotonic improvements that EM provides. Their key practical advantage lies in mini-batch processing, which scales well to large datasets and modern parallel hardware. The tractability of marginal log-likelihoods of traditional click models makes gradient optimization particularly attractive, as automatic differentiation eliminates the need for manual E and M step derivations while potentially offering computational efficiency gains.

Lastly, the relationship between EM and gradient methods runs deeper than their shared objective. A fundamental property links the two approaches: the gradient of the expected complete-data log-likelihood ($\mathcal{Q}$-function), evaluated at the current parameter estimates, is equal to the gradient of the marginal log-likelihood [158]:

$$\nabla \mathcal{Q}(\theta, \gamma \mid \theta^{(t)}, \gamma^{(t)})\Big|_{\theta=\theta^{(t)},\gamma=\gamma^{(t)}} = \nabla \mathcal{L}(\theta, \gamma)\Big|_{\theta=\theta^{(t)},\gamma=\gamma^{(t)}}. \tag{5.10}$$

In fact, EM can be viewed as gradient ascent with an adaptive, parameter-dependent step size [158]. We verify this equality explicitly for the PBM (Appendix 5.B). The gradient of the auxiliary function $\mathcal{Q}$ with respect to γ_d, evaluated at $\theta_k = \theta_k^{(t)}$ and $\gamma_d = \gamma_d^{(t)}$, is:

$$\frac{\partial \mathcal{Q}\left(\theta_k, \gamma_d \mid \theta_k, \gamma_d\right)}{\partial \gamma_d} = \sum_{(d', k, c) \in \mathcal{D}, d'=d} \left(\frac{c}{\gamma_d} - \frac{(1-c)\theta_k}{1 - \theta_k \gamma_d} \right) = \frac{\partial \mathcal{L}}{\partial \gamma_d}. \tag{5.11}$$

While we verify Eq. 5.10 only explicitly for the PBM for brevity, the fundamental property holds for any click model optimized using EM, including more complex models like the DCM, UBM, and DBN [158].

This connection has important practical consequences. If one implements an EM algorithm using the most recent model parameters in the E-step and performs only a single gradient update on a batch of data during the M-step, EM effectively becomes direct gradient-based maximization of the marginal log-likelihood. For instance, Google's TensorFlow Ranking library implements an EM-based objective for the PBM [184].[8] However, their gradient-based optimization of this objective produces the same gradient for a single batch of data as directly minimizing the simpler binary cross-entropy loss in Eq. 5.2. This demonstrates that some industry practitioners have implicitly adopted direct gradient-based optimization of click models, while framing it as EM.

Nevertheless, we end this comparison by emphasizing that classical implementations of both approaches often differ: EM typically performs full maximization in each M-step or aggregates the entire dataset in the E-step, while gradient-based methods exploit stochasticity and adaptive optimization techniques.

Our analysis demonstrates that gradient-based optimization is a theoretically sound alternative to EM for PGM-based click models with tractable marginal likelihoods. This insight motivates CLAX, a JAX-based library for flexible neural parameterization and scalable optimization. We introduce the design of CLAX in the following section.

5.4 An Overview of CLAX

We designed CLAX around three principles: (i) direct and numerically stable log-likelihood optimization to replace EM; (ii) decoupling model logic and parameterization for flexible model composition; and (iii) speed and memory-efficiency to support scale. Below, we give an overview of the CLAX API and detail decisions that enable flexible parameterization and scale. We cover numerical stability separately in Section 5.5.

[8] https://www.tensorflow.org/ranking/api_docs/python/tfr/keras/losses/ClickEMLoss

Listing 5.2: The CLAX model API.

```
batch = {
    "positions": [[1, 2, 3, ...]],
    "query_doc_ids": [[101, 205, 847, ...]],
    "clicks": [[0., 1., 0., ...]],
    "mask": [[True, True, True, ...]],
}

loss = model.compute_loss(batch)
log_probs = model.predict_clicks(batch)
cond_log_probs = model.predict_conditional_clicks(batch)
relevance_scores = model.predict_relevance(batch)
output = model.sample(batch, rngs=nnx.Rngs(42))
```

5.4.1 The CLAX model API

All click models in CLAX share a unified interface of five methods that accept a batch of data, as demonstrated in Listing 5.2 above. A batch in CLAX is a Python dictionary of 2D NumPy arrays of shape (batch size, max. positions). By default, CLAX expects an array of document positions, query-document-ids, clicks, and a binary mask. The variables and their names depend on the model parameterization and can be changed.

We require all queries within a batch to be padded to the same shape. This allows vectorizing operations across variable-length user sessions and reduces JIT recompilation when JAX encounters arrays of different lengths. A binary mask indicates which query-document pairs a model or metric should use per batch, a common pattern in Jax [18, 94]. Each CLAX model implements five methods:

- **`compute_loss(...)`** Computes the training objective, typically the negative log-likelihood of observed clicks, but models may implement custom loss functions.
- **`predict_clicks(...)`** Returns log-probabilities of a click for each document d at rank k: $\hat{c} = P(C = 1 \mid d, k)$.
- **`predict_conditional_clicks(...)`** Returns click log-probabilities conditioned on previous clicks in the session: $\hat{c} = P(C = 1 \mid d, k, c_{<k})$.
- **`predict_relevance(...)`** Returns ranking scores for documents, typically the document attractiveness, though some models (e.g., the DBN) rank by attractiveness and satisfaction.
- **`sample(...)`** Generates click sequences for the current batch and also returns all latent variables (examination, attractiveness, satisfaction) sampled in the process.

5.4.2 Flexible parameterization

By decoupling the structure of each click model, i.e., how variables interact with each other, from the actual parameterization, we allow flexible composition of models.

Listing 5.3: Adding hashing and baseline correction to a CCM.

```
model = ClickChainModel(
    attraction=EmbeddingParameterConfig(
        use_feature="query_doc_ids",
        parameters=100_000_000,
        add_baseline=True,
        embedding_fn=partial(
            HashEmbedding,
            compression_ratio=10
        ),
    ),
    rngs=nnx.Rngs(42),
)
```

CLAX supports embeddings, deep neural networks, and custom Flax models, enabling researchers and practitioners to adopt parameters to their specific use case. The following is a brief overview of parameterization options in CLAX.

Embeddings

Click models commonly allocate separate parameters for different model components. For instance, examination parameters across positions or distinct attractiveness parameters per query-document pair. Therefore, CLAX uses embedding tables by default. Consider the UBM [56] in Listing 5.1. By default, the model allocates 100M embeddings for query-document attractiveness and a table of examination parameters. CLAX offers two extensions beyond traditional embedding tables that enable more accurate click predictions and scaling to larger datasets: baseline corrections and compression. Additionally, CLAX supports feature-based models as an alternative to embedding tables, which we cover afterward.

Baseline correction

Learning strictly separated parameters for attractiveness is challenging when many query-document pairs rarely occur. Therefore, CLAX optionally adds a shared baseline parameter to all embeddings, so embeddings encode their offset from the global value rather than absolute values. New parameters start at the baseline and gradually adapt with more observations, which improves click prediction on long-tailed data.

Compression

Embedding tables can rapidly expand, exhausting memory and affecting computational efficiency. Most deep learning frameworks compute gradients for entire embedding tables, even though only the embeddings used in the current batch have non-zero gradients.[9] While GPUs mitigate this inefficiency through parallel computation, CPU-based training can slow down drastically as embedding tables grow. Thus, CLAX

[9] PyTorch solves this problem with sparse embedding tables and specialized optimizers, e.g., https://docs.pytorch.org/docs/stable/generated/torch.optim.SparseAdam.html. Sparse embeddings in Jax are still under construction at the time of writing: https://github.com/jax-ml/jax-tpu-embedding

provides two embedding compressions: (i) The hashing-trick [188] maps multiple indices to the same embedding using hash functions, reducing table size at the cost of hash collisions. (ii) The quotient-remainder trick [163] splits each embedding into components from two smaller tables based on the quotient and remainder of the embedding index. The final representation is a combination of both embeddings, reducing memory usage and embedding collisions. Listing 5.3 shows how to configure the hashing-trick for the Click Chain Model (CCM). A compression ratio of ten means that the method will hash 100M embeddings down to 10M embedding parameters. Section 5.7.2 evaluates both compression techniques and demonstrates training on datasets with billions of query-document pairs on a single GPU.

Feature-based models

In many cases, allocating separate embedding parameters for models might not be optimal, e.g., when a dataset is too sparse. Instead, we may want to generalize over shared feature representations, like two-tower models that use feature representations for examination and attractiveness parameters [72, 195, 202]. CLAX supports easy configuration of any traditional click model to use features, with built-in support for linear layers, deep-neural networks, and DeepCrossV2 networks, which explicitly learn higher-order feature interactions [182]. Listing 5.4 configures a two-tower model using a linear combination of bias features to predict examination and a DeepCrossV2 network to predict relevance from 136-dimensional feature vectors:

Listing 5.4: Building a two-tower model in CLAX.

```
model = PositionBasedModel(
    examination=LinearParameterConfig(
        use_feature="bias_features",
        features=8,
    ),
    attraction=DeepCrossParameterConfig(
        use_feature="query_doc_features",
        features=136,
        cross_layers=2,
        deep_layers=2,
        combination=Combination.STACKED,
    ),
    rngs=nnx.Rngs(42),
)
```

Lastly, we note that model parameters can be any Flax module, as long as the output shape of the module matches the expectations of the click model as we demonstrate in our online repository.

5.4.3 Mixture models

The power of modular design and gradient-based optimization becomes more apparent when exploring meta-modeling approaches that combine multiple click models to capture diverse user behaviors. A prominent example is the MixtureEM method proposed by Yan et al. [195], which uses the EM algorithm to learn a distribution over multiple click models, capturing different user behaviors across sessions. This approach

Listing 5.5: A mixture model with parameter sharing.

```
rngs = nnx.Rngs(42)
attraction = EmbeddingParameter(
    EmbeddingParameterConfig(
        use_feature="query_doc_ids",
        parameters=100_000_000
    ),
    rngs=rngs,
)
pbm = PositionBasedModel(
    attraction=attraction, positions=10, rngs=rngs
)
dbn = DynamicBayesianNetwork(
    attraction=attraction, positions=10, rngs=rngs
)
model = MixtureModel(models=[pbm, dbn])
```

recognizes that users may exhibit distinct browsing patterns across queries. The original MixtureEM approach alternates between computing posterior probabilities for assigning each session to a model based on observed clicks, then training individual models with weighted losses based on these posteriors. While MixtureEM can effectively train an unbiased ranker, the posterior computation requires observed clicks, meaning it cannot be used to predict clicks on unseen rankings [195].

CLAX offers a mixture model that extends the idea of Yan et al. [195] and can be used like any other CLAX model. Our mixture model combines M different click models, where each model $m \in M$ has a learnable prior probability $P(m)$ and produces a session-level log-loss $\mathcal{LL}_m(s)$. The loss of the mixture model is:

$$\mathcal{LL}_{\text{mixture}}(s) = -\log\left(\sum_{m \in M} P(m)\exp(-\mathcal{LL}_m(s)/\tau)\right), \tag{5.12}$$

where τ is a temperature parameter controlling how much the mixture concentrates on the best-fitting models for each session. The learnable priors $P(m)$ are jointly optimized with the marginal log-likelihood of each model using gradient descent. The resulting model enables click prediction for new sessions without requiring observed clicks to compute posteriors and fully uses end-to-end gradient-based optimization. Listing 5.5 shows how to learn a mixture distribution over a PBM and DBN model: Note that Yan et al. [195] share parameters between different click models, which is not necessary in CLAX, but easy to do as we can supply the same parameter to both models, as shown in Listing 5.5. We evaluate a mixture model as part of our experiments in Section 5.7.3.

5.4.4 Evaluation

More critical than training models is evaluating click models. Typically, we assess two main aspects of click models [37, 69]: a model's ability to predict clicks, which is evaluated on a hold-out test set of clicks, and the model's ability to rank documents, which is assessed against relevance judgments from expert annotators. In the following, we cover the evaluation metrics implemented in CLAX.

Log-likelihood

The most common metric for click prediction is the log-likelihood, measuring how well a model fits observed clicks [37]:

$$\mathrm{LL}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(d,k,c)\in\mathcal{D}} \Big[c \log \hat{c} + (1-c) \log (1-\hat{c}) \Big], \tag{5.13}$$

where $\hat{c} = P(C = 1 \mid d, k, c_{<k})$ are a model's click predictions, conditioned on clicks observed before the current rank k. Log-likelihood values are negative, with higher values (closer to zero) indicating better model fit.

Perplexity

Perplexity offers a more intuitive interpretation than log-likelihood [37, 56]. It measures how *surprised* a model is by the observed data, with a lower value indicating a better model fit. Intuitively, it represents the weighted average number of choices a model is considering. Perfect predictions yield a perplexity of 1, while random guessing for binary outcomes gives a perplexity of 2; the model is as uncertain as a coin flip. Perplexity is defined as:

$$\mathrm{PPL}(\mathcal{D}) = 2^{-\frac{1}{|\mathcal{D}|} \sum_{(d,k,c)\in\mathcal{D}} [c \log_2 \hat{c} + (1-c) \log_2 (1-\hat{c})]}. \tag{5.14}$$

Perplexity metrics differ based on how the click prediction $\hat{c}$ is calculated: (i) Conditional perplexity uses $\hat{c} = P(C = 1 \mid d, k, c_{<k})$, where predictions can incorporate clicks observed at previous ranks. (ii) Unconditional perplexity uses $\hat{c} = P(C = 1 \mid d, k)$, without considering clicks from the current session. This distinction is important, since some click models adapt their predictions based on clicks in the current sessions (such as [56]). Conditional perplexity measures how well a model fits an observed dataset, while unconditional perplexity reflects how accurately a model can predict clicks on a completely unseen list of documents. Note that conditional perplexity assumes top-down browsing behavior; when this assumption is violated, unconditional perplexity is preferable. CLAX implements all three standard click prediction metrics with support for global and rank-based averaging. Similar to the model API, all metrics handle batched inputs with a binary mask indicating which input document are not padding. Our API follows the NNX metrics API[10] and supports input routing, meaning all metrics can be updated at once with each metric automatically extracting the arguments it requires, as shown in Listing 5.6.

Ranking metrics

A second aspect commonly evaluated of click models is their ranking performance [37], which (in web search) is typically assessed against expert-annotated relevance labels using metrics such as discounted cumulative gain (DCG), mean reciprocal rank (MRR), or average precision (AP). Instead of reimplementing these metrics, CLAX supports integrating ranking metrics from Rax, the most prevalent JAX-based library for learning-to-rank [94]. Listing 5.7 shows how Rax-based ranking metrics can be used in CLAX.

[10] https://flax.readthedocs.io/en/latest/api_reference/flax.nnx/training/metrics.html

Listing 5.6: Computing click metrics.

```
metrics = MultiMetric({
    "ll": LogLikelihood(),
    "ppl": Perplexity(),
    "cond_ppl": ConditionalPerplexity(),
})


metrics.update(
    log_probs=log_probs,
    conditional_log_probs=cond_log_probs,
    clicks=clicks,
    where=mask,
)

results = metrics.compute()
rank_results = metrics.compute_per_rank()
```

Listing 5.7: Support for Rax-based ranking metrics.

```
metrics = MultiMetric({
    "dcg@10": RaxMetric(rax.dcg_metric, top_n=10),
    "mrr@10": RaxMetric(rax.mrr_metric, top_n=10),
})

metrics.update(scores=scores, labels=labels, where=mask)
results = metrics.compute()
```

Following this overview of CLAX, we introduce the basic techniques used to achieve numerical stability in this work.

5.5 Numerical Stability

Optimizing complex likelihood expressions using gradient descent requires attention to numerical stability. The marginal likelihoods of many common click models contain products of small probabilities, which can lead to numerical underflow in finite-precision computer arithmetic [65, 100]. Below, we cover the techniques CLAX uses to stabilize complex likelihood expressions by performing all probability computations in log-space.

5.5.1 Multiplication

By moving to log-probabilities, products of probabilities simplify to sums (and division to subtraction):

$$\log\left(\prod_{i=1}^{n} p_i\right) = \sum_{i=1}^{n} \log p_i, \tag{5.15}$$

which essentially eliminates the concern of numerical underflow when multiplying small probabilities.

5.5.2 Addition

While multiplication becomes stable (and faster) in log-space, the addition of probabilities becomes more complicated as it requires first exponentiating log probabilities. This reintroduces the problems we seek to avoid, as exponentiating large positive inputs lead to overflow and exponentiating large negative inputs lead to underflow. The standard solution is to avoid large inputs to the $\exp(\cdot)$ operation via the log-sum-exp trick [15]:

$$\texttt{log\_sum\_exp}(a) = a_{\text{max}} + \log\left(\sum_{i=1}^{n} \exp(a_i - a_{\text{max}})\right), \tag{5.16}$$

where $a = (a_1, \ldots, a_n)$ is a vector of log values and $a_{\text{max}} = \max_i(a_i)$ is the maximum input value. The trick is prevalent in probabilistic modeling, and we also use it to transform the output logits of neural networks $x \in \mathbb{R}$ to log-probabilities by implementing numerically stable versions of the log-sigmoid functions:

$$\begin{aligned} \log(\sigma(x)) &= -\texttt{log\_sum\_exp}([0, -x]), \text{ or} \\ \log(1 - \sigma(x)) &= -\texttt{log\_sum\_exp}([0, x]). \end{aligned} \tag{5.17}$$

5.5.3 Complements and cancellation

Sometimes we need to compute the log of a complement $\log(1 - p)$, e.g., in the binary-cross entropy loss or when computing log-posteriors in the DBN [25]. Performing this step directly from log-probability $\log p$ requires computing: $\log(1 - \exp(\log p))$. This expression is numerically unstable in two ways: (i) underflow: when p is very small, $\log p$ is very negative, causing $\exp(\log p)$ to underflow to zero; and (ii) catastrophic cancellation: when $p \approx 1$, we have $\exp(\log p) \approx 1$, making $1 - \exp(\log p) \approx 0$, since subtracting nearly equal floating point numbers leads to a loss of precision [65]. Therefore, we compute $\texttt{log1mexp}(x)$ as proposed in [123] and adopted by major frameworks such as TensorFlow[11] and JAX.[12] Mächler [123] proposes a piecewise approximation that switches between two stable expressions that are precise in different input ranges.[13] For a log-probability $a \in \mathbb{R}, a \leq 0$:

$$\texttt{log1mexp}(a) = \begin{cases} \log(-\text{expm1}(a)) & \text{if } a > -\log(2) \\ \text{log1p}(-\exp(a)) & \text{if } a \leq -\log(2). \end{cases} \tag{5.18}$$

The implementation relies on the standard functions $\text{log1p}(x)$, which accurately computes $\log(1 + x)$, and $\text{expm1}(x)$, which accurately computes $\exp(x) - 1$, to avoid catastrophic cancellation.

To summarize, CLAX performs all probability computations in log space for increased numerical stability, avoiding underflow and overflow as well as catastrophic cancellation. We list all implemented log-likelihoods in Appendix 5.A and their corresponding implementation can be found in our code repository.[14]

[11] https://www.tensorflow.org/probability/api_docs/python/tfp/math/log1mexp

[12] https://docs.jax.dev/en/latest/_autosummary/jax.nn.log1mexp.html

[13] Interested readers can find the motivation behind the switching point $\log(2)$ under [123, Section 2].

[14] https://github.com/philipphager/clax

5.6 Experimental Setup

We conduct experiments in three settings to evaluate CLAX. First, we compare CLAX models to EM-based counterparts from PyClick. Second, we scale CLAX to large datasets and investigate the effects of embedding compression. Third, we evaluate CLAX models as unbiased ranking models. Next, we introduce the basic setup shared across all our experiments.

5.6.1 Datasets

We use two real-world datasets of user interactions with search engines: The WSCD-2012 dataset by Yandex is a foundational benchmark in click modeling [37, 162]. It contains 146,278,823 user sessions and 346,711,929 unique query-document pairs. The dataset provides query and document identifiers without additional document features, allowing only for a direct comparison of embedding-based click models. We generate a unique identifier for each query-document combination as the only preprocessing step.

The Baidu-ULTR dataset is the largest publicly-available real-world dataset for unbiased learning-to-rank, comprising over 1.2 billion user sessions and a test set of 397,572 annotated query-document pairs [209]. This scale allows us to verify CLAX's scalability and ranking performance. We employ hashing to generate query-document IDs from query IDs and document URLs, yielding 2,147,483,647 unique identifiers. We are the first to train on the entire Baidu-ULTR dataset, rather than just a subset [33, 80, 116, 209, 211]. For ranking performance comparison, we use the subset of Baidu-ULTR created by Hager et al. [80] with pre-computed 768-dimensional MonoBERT features for 2,372,947 sessions.[15] We publish all pre-processed datasets and highly efficient custom dataloaders using Apache Parquet under: https://huggingface.co/datasets/philipphager/clax-datasets.

5.6.2 Implementation

All CLAX experiments use the default trainer with the AdamW optimizer [121] (learning rate 0.003, weight decay 0.0001) over 100 epochs, stopping early after the first epoch without improvement of the validation loss. Beyond our preliminary experiments, hyperparameters should be tuned per model and dataset. All experiments run over three dataset splits and random seeds, we plot bootstrapped 95% confidence intervals. CLAX experiments use a single NVIDIA RTX A6000 GPU (48GB RAM) and PyClick experiments use an Intel Xeon Gold 5118 CPU (2.30GHz). As recommended, we run PyClick on the PyPy interpreter with JIT-compilation. To ensure a fair comparison with CLAX, we adjust PyClick's parameter initialization from $\frac{1}{2}$ to $\frac{1}{9}$ to better reflect the mean CTR on WSCD-2012, improving click prediction on long-tailed items. We publish all experimental configurations including detailed data splits and baselines in our code repository: https://github.com/philipphager/clax.

[15] https://huggingface.co/datasets/philipphager/baidu-ultr_baidu-mlm-ctr

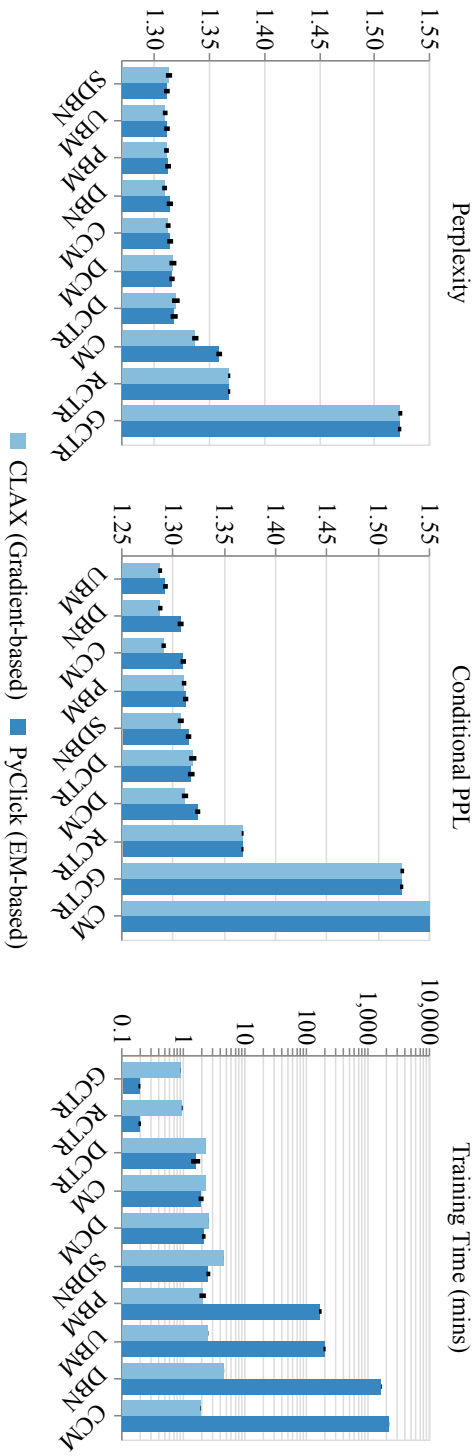


Figure 5.1: CLAX matches or exceeds the click predictions of PyClick over three folds of 10M training sessions on WSCD-2012.

5.7 Results

5.7.1 Comparing EM and gradient-based optimization

We validate CLAX against PyClick, the prevailing click modeling library, to ensure our implementation produces equivalent click predictions. Due to the scalability limitations of PyClick, we train on three folds of 10M user sessions from the WSCD-2012 dataset, using 5M sessions each for validation and testing. Figure 5.1 compares the click prediction performance and training times across both libraries. First, we note that the (unconditional) perplexity matches closely between the two libraries. For conditional perplexity, simple models, such as the PBM and CTR-based models, achieve identical performance. Surprisingly, CLAX sometimes achieves better conditional perplexity despite both libraries optimizing the same objectives. After further investigation, we attribute this improvement to CLAX's enhanced numerical stability, as we find improved click predictions at lower ranks. Regarding training time, PyClick excels for MLE-based models, which just require counting. Training the EM-based models on 10M sessions, however, requires from 172 minutes (PBM) to 36 hours (CCM).[16] In contrast, all CLAX models complete training in under 5 minutes. To summarize, CLAX matches PyClick's unconditional click prediction performance while matching or exceeding conditional prediction accuracy, potentially leading to different conclusions about optimal model selection for specific datasets.

5.7.2 Scaling up CLAX

Next, we evaluate CLAX on large datasets. To begin, we evaluate the hashing trick and quotient-remainder embedding compressions. To assess how compression might change our conclusions about model fit, we compare the obtained model ranking when training with compression versus training on the full embedding table. We train on WSCD-2012 using three splits of 80M training sessions and evaluate five compression ratios, reducing the full embedding table with 346M entries by factors of 2× to 1,000×.

Figure 5.2 shows the resulting Kendall's tau rank correlation when sorting models by their click prediction performance, trained with compression, against the ranking obtained when training without compression. We observe that both compression methods behave similarly, maintaining remarkably high correlation up to very high compression ratios of 10–100×. Unconditional perplexity is more susceptible to compression. However, as Figure 5.1 reveals, many models perform similarly on this dataset, so small changes can alter rankings while preserving the overall conclusion that many models are nearly equivalent. Note that compression reduces overall click prediction performance (higher perplexity). As some CTR baselines, such as the RCTR and GCTR, do not use compression, comparing compressed and uncompressed models can lead to wrong conclusions at high compression rates. We observe that, beyond reducing the memory footprint, compression also decreases the average training time from 14 minutes for uncompressed models to around four minutes for 10x compression and higher. Secondly, we evaluate scale by training CLAX models using the hashing-trick

[16] Note that the DCM in PyClick is actually a simplified DCM (SDCM) to use faster MLE while CLAX implements the original latent-variable version [71].

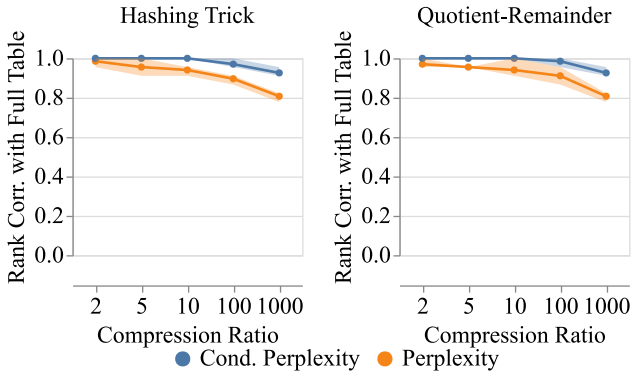


Figure 5.2: Kendall's τ between ranking models trained with and without embedding compression on WSCD-2012.

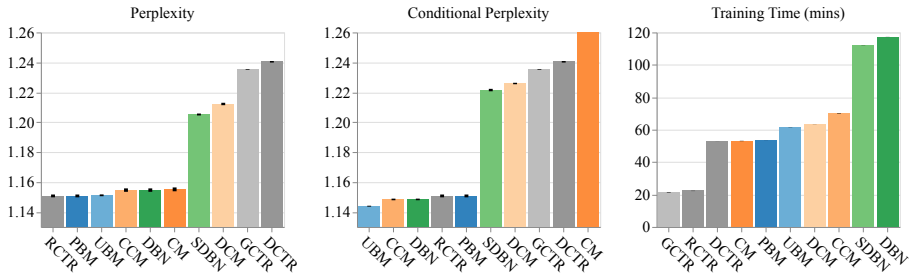


Figure 5.3: Embedding-based CLAX models on the Baidu-ULTR dataset (three folds of 800M / 200M / 200M sessions for training, validation, and testing) [209]. All models complete training in under 2 hours using the hashing-trick with 10x compression.

with 10x compression on three folds of the full Baidu-ULTR dataset containing over 1B user sessions. Figure 5.3 shows the resulting models, all trained in under 2 hours.

5.7.3 Generalizing over features

Lastly, we parameterize CLAX's attractiveness and satisfaction parameters with a deep-cross network to investigate click prediction and ranking performance when generalizing over query-document features. Figure 5.4 shows the results on the Baidu-ULTR-UVA subset [80]. While click prediction performance remains similar to that of embedding-based training, the performance gap between individual models narrows considerably when generalizing over features, leading to different conclusions about model relationships. For ranking performance, we focus on the DCTR model (corresponding to a naive model without bias correction in unbiased learning-to-rank) and PBM (corresponding to a two-tower model). Ranking performance on Baidu-ULTR does not directly correlate with click prediction performance, a known problem on this dataset [80]. Nevertheless, cascade-based models achieve strong ranking performance, comparable to listwise LTR

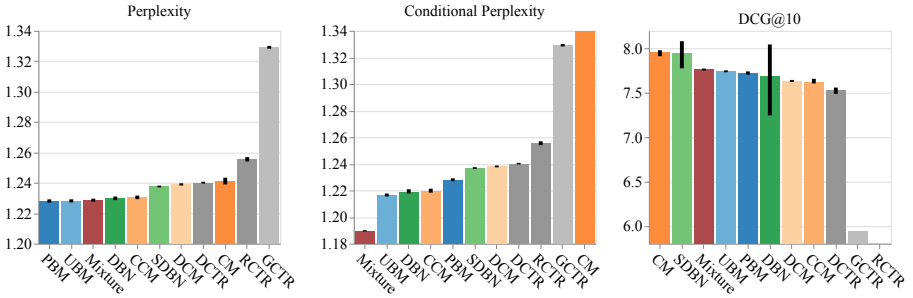


Figure 5.4: CLAX models generalizing over BERT features on the Baidu-ULTR-UVA dataset [80] using a deep-cross network achieve strong ranking performance and a different model fit compared to embedding-based models.

loss functions trained in previous work [80, Figure 3]. Our results suggest that complex click models can be effective ranking models in real-world settings.

Finally, we evaluate the effectiveness of our mixture model in Figure 5.4, which combines a PBM, DCTR, and GCTR model following the setup of Yan et al. [195] but excluding the RCTR as it cannot be applied to the Baidu-ULTR test. The mixture model achieves better click prediction and ranking performance than each individual component of the mixture model.

5.8 Conclusion

In this chapter, we investigate **RQ4** on whether gradient-based optimization can effectively replace expectation maximization for training PGM-based click models. We answer this question affirmatively. Using the PBM as an example, we demonstrate that gradient-based optimization is a valid alternative to EM when the model's log-likelihood is tractable and show its theoretical connection to online EM methods, when the expectation step uses the latest parameters, and performs a single gradient step during maximization.

Based on this theoretical foundation, we introduce CLAX, a JAX-based neural click modeling library enabling end-to-end gradient-based optimization for PGM-based click models. CLAX replaces EM with direct gradient-based optimization and implements ten established click models in a numerically stable manner. The framework achieves comparable click prediction performance to EM-based optimization while providing an order-of-magnitude speedup over established implementations at scale, training and evaluating on over 1B user sessions in $\approx$ 2 hours on a single GPU.

CLAX's modular design decouples model logic from parameterization, supporting embeddings, deep networks, and custom modules, with embedding compression techniques enabling the framework to scale to billions of query-document pairs. This flexibility serves both industry practitioners seeking to understand user behavior at scale and researchers developing new click models.

Beyond scalability, we evaluate more complex click models in a learning-to-rank setting where they generalize over query-document features. Our experiments on a

subset of the Baidu-ULTR dataset [80] show that neural parameterizations of complex PGMs and mixture models can achieve higher ranking performance on expert annotations than widely-used two-tower models. However, we also observe that different models excel at click prediction and ranking on Baidu-ULTR, a phenomenon warranting further investigation.

Future work may use CLAX to create new sophisticated PGM-based neural click models that explore user signals beyond clicks [209] or explore new methods to combine click models to account for mixtures of user behaviors [195]. Finally, the implementation introduced in this chapter currently lacks support for sparse embeddings, which can negatively impact performance on CPUs. Future iterations of CLAX will add support for sparse embeddings and distributed training across GPUs.

5.9 Reproducibility

CLAX is openly available under the MIT license.

- The framework, all experiments, and documentation are published at: https://github.com/philipphager/clax.
- A pre-processed version of the Yandex WSCD-2012 dataset and two pre-processed versions of the Baidu-ULTR dataset with highly efficient custom dataloaders are available at: https://huggingface.co/datasets/philipphager/clax-datasets.

5.A Models

We introduce the ten click models implemented in CLAX following the standard work of Chuklin et al. [37, Table 3.1] and their click log-probabilities. If we do not explicitly state a conditional click probability, unconditional and conditional click probabilities are equal for the specific model.

5.A.1 Global CTR model (GCTR)

The global CTR (GCTR) model, sometimes called the random click model, predicts a single average CTR across all documents. It serves as a simple baseline that any click model should surpass:

$$\log P(C = 1 \mid d, k) = \log \rho. \tag{5.19}$$

5.A.2 Rank-based CTR model (RCTR)

The rank-based CTR (RCTR) model assumes that click probability depends only on the rank k of a document and not its content. The model predicts the average CTR for all documents displayed at the same rank, treating them as equally attractive:

$$\log P(C = 1 \mid d, k) = \log \theta_k. \tag{5.20}$$

5.A.3 Document-based CTR model (DCTR)

The document-based CTR (DCTR) model assumes that clicks depend solely on a document and not on its position in the ranking:

$$\log P(C = 1 \mid d, k) = \log \gamma_d. \tag{5.21}$$

5.A.4 Position-based model (PBM)

The PBM assumes that clicks occurs only if a user first examines the result at rank k (with probability θ_k), and if the displayed document is attractive (γ_d):

$$\log P(C = 1 \mid d, k) = \log \theta_k + \log \gamma_d. \tag{5.22}$$

5.A.5 Cascade model (CM)

The cascade model (CM) assumes that users scan results from top to bottom, click on the first attractive document they find, and then immediately stop their search. The probability of a click at rank k depends on the displayed document d being attractive (γ_d) and all preceding documents being unattractive:

$$\log P(C = 1 \mid d, k) = \log \gamma_d + \sum_{i=1}^{k-1} \log(1 - \gamma_{d_i}). \tag{5.23}$$

Note that the cascade model can only explain a single click per list. All other documents after the first click, by definition, have a click probability of 0. To avoid a log-likelihood of $-\infty$ in our conditional click predictions, we follow the common practice to assign a very small default click probability to all documents following a click [37]:

$$\log P(C = 1 \mid d, k, c_{<k}) = \begin{cases} \log \gamma_d & \text{if } \sum_{i=1}^{k-1} c_i = 0 \\ \text{min\_log\_prob} & \text{otherwise.} \end{cases} \tag{5.24}$$

5.A.6 User browsing model (UBM)

The user browsing model (UBM) extends the PBM by assuming that the probability of examination at position k depends also on the position of the last clicked document k'. This is most easily demonstrated in the conditional log-probability of click:

$$\log P(C = 1 \mid d, k, c_{<k}) = \log \theta_{k,k'} + \log \gamma_d, \tag{5.25}$$

where k is the position of the current document and k' the position of the previously last clicked document. While conditional click probabilities are very simple, predicting clicks on a new list of documents is harder under the UBM, since it requires marginalizing over all possible last click positions $i < k$ before our current position:

$$\log P(C = 1 \mid d, k) = \\ \log \left(\sum_{i=0}^{k-1} P(C = 1 \mid d_i, i) \cdot \left(\prod_{j=i+1}^{k-1} (1 - \theta_{j,i} \gamma_{d_j}) \right) \theta_{k,i} \gamma_d \right). \tag{5.26}$$

Each term in the sum represents a path to the current document: the probability of clicking at a previous rank i, then not clicking on anything until rank k, and finally examining and clicking the document at rank k given i was the last clicked position.

5.A.7 Dependent click model (DCM)

The dependent click model (DCM) is an extension of the cascade model to explain multiple clicks in a single ranking. The DCM assumes that users examine a list from top to bottom, click on relevant items, and after clicking have a rank-dependent probability λ_k to continue browsing:

$$\begin{aligned} \log P(C = 1 \mid d, k) &= \log(\epsilon_k) + \log(\gamma_d) \\ \log(\epsilon_{k+1}) &= \log(\epsilon_k) + \log(\gamma_{d_k} \lambda_k + (1 - \gamma_{d_k})). \end{aligned} \tag{5.27}$$

When conditioning on observed clicks, the examination probability changes based on the actions in the current session:

$$\begin{aligned} \log P(C = 1 \mid d, k, c_{<k}) &= \log(\epsilon_k) + \log(\gamma_d) \\ \log(\epsilon_{k+1}) &= \log \left(c_k \lambda_k + (1 - c_k) \frac{(1 - \gamma_{d_k}) \epsilon_k}{1 - \gamma_{d_k} \epsilon_k} \right). \end{aligned} \tag{5.28}$$

If a user clicks on a document, they continue to the next rank with probability λ_k and if they do not click, we calculate the posterior probability of examining the next rank given that we observed no click using Bayes' rule.

5.A.8 Click chain model (CCM)

The click chain model (CCM) is an extension of the DCM, assuming a total of three continuation scenarios that do not only explain continuation after clicking a document but also allow users to abandon a session without any clicks. First, τ_1 is the probability of a user continuing to the next document after not clicking on the current document. Second, if the user clicks on the current document but is not satisfied, τ_2 is the probability of the user continuing to the next position. And lastly, τ_3 is the probability that a user clicks on the current item, finds it satisfying, but still wants to continue to the next document:

$$\begin{aligned} \log P(C = 1 \mid d, k) &= \log(\gamma_d) + \log(\epsilon_k) \\ \log(\epsilon_{k+1}) &= \log(\epsilon_k) \\ &\quad + \log\left(\gamma_{d_k}((1 - \gamma_{d_k})\tau_2 + \gamma_{d_k}\tau_3\right) \\ &\quad + (1 - \gamma_{d_k})\tau_1\big)\,. \end{aligned} \tag{5.29}$$

When conditioning on the observed clicks, the update rule for the examination probability changes based on the user's action at the current rank. If a click occurred, we compute continuation based on satisfaction (equal to attractiveness γ_d) and the continuation probabilities τ_2 and τ_3. If no click was observed, we compute the posterior log probability of continuing to the next rank:

$$\begin{aligned} \log P(C = 1 \mid d, k, c_{<k}) &= \log(\gamma_d) + \log(\epsilon_k) \\ \log(\epsilon_{k+1}) &= c_k \left[\log\left(\gamma_{d_k}\tau_3 + (1 - \gamma_{d_k})\tau_2\right)\right] \\ &\quad + (1 - c_k)\left[\log(1 - \gamma_{d_k}) + \log(\epsilon_k)\right. \\ &\quad \left. + \log(\tau_1) - \log(1 - \gamma_{d_k}\epsilon_k)\right]. \end{aligned} \tag{5.30}$$

5.A.9 Dynamic Bayesian network (DBN)

The dynamic Bayesian network (DBN) model of [25] separates the concepts of a document being attractive (γ_d) and being satisfying (σ_d). A user stops their search only if they click on an attractive document and are satisfied by it. If they do not click or are not satisfied by the clicked document, they continue browsing with a global continuation probability λ:

$$\begin{aligned} \log P(C = 1 \mid d, k) &= \log(\gamma_d) + \log(\epsilon_k) \\ \log(\epsilon_{k+1}) &= \log(\epsilon_k) + \log(\lambda) + \log(1 - \gamma_{d_k}\sigma_{d_k}). \end{aligned} \tag{5.31}$$

The conditional click probability again takes the user's actions in the current session into account. If a click was observed, we compute the probability of continuation based on satisfaction. If no click was observed, we compute the posterior probability of continuing to the next item:

$$\begin{aligned} \log P(C = 1 \mid d, k, c_{<k}) &= \log(\gamma_d) + \log(\epsilon_k) \\ \log(\epsilon_{k+1}) &= \log(\lambda) + c_k \left[\log(1 - \sigma_{d_k})\right] \\ &\quad + (1 - c_k)\left[\log(1 - \gamma_{d_k}) + \log(\epsilon_k)\right. \\ &\quad \left. - \log(1 - \gamma_{d_k}\epsilon_k)\right]. \end{aligned} \tag{5.32}$$

5.B Proof of PBM equality

Proof that the gradient of the expected complete-data log-likelihood ($\mathcal{Q}$-function), evaluated at the current parameter estimates, is equal to the gradient of the marginal log-likelihood [158] that is being used during gradient-based optimization:

$$\begin{aligned}
\frac{\partial \mathcal{Q}\left(\theta_k, \gamma_d \mid \theta_k, \gamma_d\right)}{\partial \gamma_d} &= \sum_{(d', k, c) \in \mathcal{D}, d'=d} \frac{\hat{a}}{\gamma_d} - \frac{1-\hat{a}}{1-\gamma_d} \\
&= \sum_{(d', k, c) \in \mathcal{D}, d'=d} \frac{1}{\gamma_d}\left[c + (1-c)\frac{(1-\theta_k)\gamma_d}{1-\theta_k\gamma_d}\right] - \\
&\qquad \frac{1}{1-\gamma_d}\left[1 - \left(c + (1-c)\frac{(1-\theta_k)\gamma_d}{1-\theta_k\gamma_d}\right)\right] \\
&= \sum_{(d', k, c) \in \mathcal{D}, d'=d} \frac{c}{\gamma_d} - \frac{(1-c)\theta_k}{1-\theta_k\gamma_d} \\
&= \frac{\partial \mathcal{L}}{\partial \gamma_d}.
\end{aligned} \tag{5.33}$$

6 Conclusions

This thesis addressed four research questions to advance neural click models and the broader field of unbiased learning-to-rank (ULTR). First, we examined the theoretical relationship between neural click models and IPS-based counterfactual ranking methods, investigating their equivalence in pointwise ranking settings. Second, we evaluated established ULTR methods on Baidu-ULTR, the largest real-world search dataset, moving beyond semi-synthetic simulations to assess bias mitigation methods in practice. Third, we investigated the conditions under which two-tower models, the most prevalent neural click models in industry, can be reliably trained from click data, analyzing both identifiability requirements and the effects of logging policies. Finally, we demonstrated that gradient-based optimization can effectively replace expectation maximization for training neural click models, enabling scalable optimization of flexible ranking models that can capture complex user behaviors. This final chapter reflects on the main findings for each research question before concluding with opportunities for future work.

6.1 Summary of Findings

RQ1 Are neural click models and inverse-propensity scoring equivalent approaches for addressing position-bias in a pointwise ranking setting?

Chapter 2, based on [79], compares the two dominant approaches for learning unbiased ranking models from position-biased clicks: inverse-propensity scoring (IPS) [13, 99, 157] and neural click models [184, 195]. We investigate their theoretical and empirical equivalence in a pointwise ranking setting under the same model of position bias, the position-based model (PBM) [37, 153]. Assuming the idealized scenario in which users behave according to the PBM and that the dataset's position bias is known, we prove theoretically that both methods optimize for unbiased document relevance when the models allocate separate parameters per query-document pair.

However, our simulation experiments also reveal small but significant empirical differences in ranking performance when generalizing across shared query-document features. Our investigation shows that neural click models weight gradient updates toward documents with higher examination probabilities, while IPS weights documents across all positions equally. We validate this finding through controlled experiments that introduce feature collisions and vary the severity of position bias.

Overall, we answer RQ1 negatively: neural click models are generally not equivalent to pointwise IPS rankers, as neural click models may be impacted by position bias when learning from shared, potentially conflicting document features.

RQ2 Do unbiased learning-to-rank methods that work well in simulation actually improve ranking performance on the largest real-world search dataset?

In Chapter 3, based on [80], we investigate whether six established unbiased learning-to-rank methods [5, 13, 90, 99, 184, 195] for position bias correction improve ranking performance on Baidu-ULTR, a large real-world search engine dataset [209]. Despite finding convincing evidence that click feedback in Baidu-ULTR is affected by position bias, we found that the compared ULTR methods do not consistently improve ranking performance over naive baselines without bias correction when evaluated against expert annotations, confirming earlier findings by Zou et al. [209] despite our use of different preprocessing, model implementations, and extensive hyperparameter tuning. The choice of ranking loss function and query-document features had a substantially greater impact on ranking than bias mitigation techniques. Thus, we answer RQ2 negatively.

While ULTR methods reliably improved click prediction, gains in click prediction did not consistently translate into improved ranking performance on expert annotations. Strikingly, all click-based approaches were outperformed by simple heuristics such as BM25 [155], even when BM25 scores were used as inputs to models trained on clicks, pointing to a fundamental disconnect between click-based and annotation-based objectives on the Baidu-ULTR dataset and warranting further research.

Our results demonstrate that applying position bias correction methods as black-box methods without a deeper user understanding does not necessarily improve ranking performance and can even degrade it, as we observed when applying IPS corrections to transformer-based cross-encoders [133]. These findings call for deeper investigation into the conditions for success and failure of ULTR methods when applied to modern neural IR methods [118] in real-world settings, research which we hope to facilitate through the release of pre-processed datasets, pre-trained models, and optimized baseline implementations.

RQ3 When can two-tower models reliably disentangle bias and relevance signals from clicks, and what role do the production systems play collecting the clicks?

Chapter 4, based on [83], investigates two-tower models, a prevalent neural click model architecture for position bias correction in industry [72, 85, 105, 195, 202]. Recent work observed that training two-tower models on clicks collected by strong logging policies (production systems that produce near-optimal rankings) degrades ranking performance, raising concerns about a fundamental tension between improving production systems and training unbiased ranking models [122, 200].

We first analyze identifiability conditions for two-tower models, the theoretical requirements for recovering bias and relevance parameters from observed clicks. Extending prior work by Chen et al. [30] that required document swaps across positions, we prove that identification can also be achieved through overlapping query-document feature distributions across ranks. However, feature-based identification assumes a continuous relevance model, sufficient overlap across bias dimensions, and correct model

specification. In practice, feature overlap decreases with feature dimensionality [40], and deep neural networks can produce large jumps in predictions despite being continuous [161]. We discuss these trade-offs, showing that while feature-based identification offers an alternative when randomization is impractical, explicit randomization via document swaps remains essential to ensure identifiability in many practical settings.

Contrary to existing work [200], we show that logging policies do not affect well-specified two-tower models beyond identifiability concerns. However, logging policies can amplify bias in misspecified models when click prediction errors systematically correlate with document position. We demonstrate this through three misspecification scenarios and show that the deteriorating performance observed in prior work stems from a flawed simulation setup rather than from an inherent confounding problem.

To address cases where model misspecification cannot be avoided, we propose a propensity-weighting scheme that reduces logging policy effects by reweighting query-document pairs inversely to their display probability. While this approach mitigates bias amplification from unequal exposure, it cannot fully eliminate biases arising from misspecification itself.

RQ4 Can gradient-based optimization replace Expectation-Maximization to enable scalable and flexible neural parameterization of PGM-based click models?

Chapter 5, based on [82], extends gradient-based optimization and neural parameterization beyond two-tower models to a broader class of established click models based on probabilistic graphical models (PGMs) [37, 109]. We demonstrate that gradient-based optimization is a viable alternative to expectation maximization (EM) [50] for click models with latent variables, provided their marginal log-likelihoods are feasible to compute. We discuss the theoretical connection between the two optimization approaches and show that gradient-based optimization is comparable to online EM methods that take a single gradient step during maximization and use the most recent parameters in the expectation step.

Based on these findings, we introduce CLAX, an open-source JAX-based framework for gradient-based optimization of PGM-based click models. CLAX implements ten established neural click models using automatic differentiation and numerically stable log-space computations. The framework design decouples model logic from parameterization, enabling flexible end-to-end optimization of embeddings, deep networks, and custom model components. Empirically, CLAX achieves comparable click-prediction performance to the established EM-based library, PyClick, while training orders of magnitude faster due to JAX's just-in-time compilation and GPU acceleration [18, 46]. To scale to datasets with billions of query-document pairs, CLAX incorporates embedding compression techniques from the deep recommender systems literature [163, 188]. We demonstrate CLAX's scalability by experimenting on the complete Baidu-ULTR dataset [209] comprising over 1 billion user sessions in $\approx$ 2 hours on a single GPU. Our theoretical and empirical comparisons between EM and gradient-based methods lead us to answer RQ4 in the affirmative.

Further, we demonstrate the flexibility of CLAX by implementing advanced neural components such as deep-cross networks [182] and mixture models that combine multiple click models to capture diverse user behaviors [195].

Lastly, evaluating CLAX on the preprocessed subset of Baidu-ULTR introduced in Chapter 3, we show that neural parameterizations of complex PGM-based click models achieve ranking performance comparable to that of strong listwise learning-to-rank methods. However, we again observe a divergence between click prediction and ranking tasks, similar to Chapter 3, as different models excel at either task on the Baidu-ULTR dataset, requiring further investigation. Still, we demonstrate that click models capturing more complex user behavior can improve ranking performance, providing practitioners with powerful and interpretable alternatives to standard two-tower models.

6.2 Future Work

Each research chapter in this thesis lists chapter-specific directions for future work in its limitations or conclusion section. Here, we step back to discuss broader avenues for advancing neural click models and unbiased learning-to-rank (ULTR) at large.

Richer models, interfaces, and feedback signals. The first avenue for advancing neural click models is a continuation of the rich history in click modeling of observing and formalizing new biases in user feedback, typically inspired by user studies [45, 97, 180] or novel datasets [5, 160, 209]. Continuing this trajectory for neural click models, future work may explore (i) implicit feedback on user interfaces beyond single-list layouts, such as grids [45, 73, 194] or dynamic feeds common in modern streaming and social media platforms [95, 192]; (ii) capturing heterogeneous user behavior across different user groups or over multi-session journeys [28]; and (iii) integrating feedback signals beyond clicks, including scrolling, skipping, or dwell time [62, 203, 209]. Especially impressions, information about which items the user scrolled into view [209], is a common type of feedback tracked in industry but underexplored in academic ULTR research, despite its potential to reduce the need to predict user examination patterns; and (iv) the development of richer user models capturing real-world user phenomena should also inform updates to the simulation setups used in ULTR research.

Recently published real-world datasets [44, 178, 209], including the datasets shared in this thesis [80, 82], should enable academic research on diverse user behaviors, complex layouts, and feedback beyond clicks.

Methodological advancements for neural click models. Several technical directions remain underexplored: (i) end-to-end gradient-based optimization enables new mixture or hierarchical model architectures that may capture complex latent-variable structures [195, 199]; (ii) Bayesian techniques for neural click models could quantify uncertainty in parameter estimation [9, 151]; (iii) for complex click models where exact inference becomes intractable, advanced techniques such as variational inference warrant further investigation [126, 179]; and (iv) developing methods that jointly optimize for calibrated click predictions and ranking performance remains an open challenge [11], particularly for applications like advertising auctions where both objectives matter, further unifying click models and traditional learning-to-rank methods.

The gap between clicks and annotations. As demonstrated multiple times in this thesis, click prediction performance does not guarantee ranking performance measured on expert annotations. This gap reflects two challenges: first, biases arising from how users interact with results (the traditional focus of ULTR), and second, the difference between user preferences and expert judgments. For the first challenge, we have pinpointed identifiability issues and model misspecification as potential culprits (Chapters 2 and 4). However, additional practical diagnostics are needed to guide practitioners in selecting an appropriate click model for a specific dataset. Taking this even further, rather than discovering user behavior by comparing an array of predefined click models, a practice problematic when no model fits well or identification fails on a given dataset [47, 48], future work could automatically discover the structure of click models from data, perhaps through causal discovery methods [57]. For the second challenge, research should further investigate the alignment between users and annotators on benchmark datasets, such as Baidu-ULTR, and when click-based objectives should be evaluated on their own terms (e.g., predicting user satisfaction) rather than as proxies for expert judgments.

Ethical and privacy concerns of implicit feedback. The field's reliance on vast amounts of user tracking raises ethical and privacy concerns. Future research should explore (i) ranking algorithms that require less tracking data or avoid centrally sharing user data altogether [103, 130]; (ii) methods preventing the inference of sensitive user attributes (such as gender or health status) from implicit feedback [89, 187]; (iii) meaningful consent mechanisms to avoid cases in which users unintentionally share their data or remain unaware of algorithmic personalization [77, 128, 174]; and (iv) better understanding when search engines go beyond capturing user preference and start actively shaping it [54, 58, 204].

Large language models as annotators. Lastly, interest in using large language models (LLMs) to collect relevance judgments has surged within the web search community [53, 59, 63, 149, 150, 168, 172]. LLM judgments have been found to correlate well with human assessments [8, 59, 149, 172] outperforming crowdworkers and sometimes matching trained experts [172], while operating at a much higher speed and at a fraction of the cost [59, 149]. Furthermore, automated judgments promise consistency, as models are not affected by fatigue, boredom, or context switching as humans are [8, 59]. They also unlock new evaluation capabilities, such as assessing content across languages and modalities (e.g., images and videos) at scale [59].

However, adopting LLM judges comes with significant challenges: LLM performance is highly sensitive to prompt phrasing [8, 172] and LLMs are vulnerable to adversarial manipulation, for instance, stuffing irrelevant documents with query keywords can change their relevance judgment [8, 12]. Additionally, LLMs exhibit systematic biases, including a tendency toward more positive ratings compared to human assessors [12, 63] and a preference for content and rankings generated by other LLMs [12, 63]. Despite these issues, LLMs are already actively being used as judges in community efforts such as the TREC 2024 RAG Track [169] and in parts of Microsoft's Bing search engine [172]. Future work should critically assess (i) novel problems LLM-based judgments introduce, including systematic biases in training search engines;

(ii) how LLM-based judgments compare to interaction-based rankers in personalized settings; (iii) and how hybrid systems could combine LLMs and human feedback [168].

6.3 The Lesson Beyond Clicks

In late April 2025, OpenAI addressed concerns about an update to their chatbot GPT-4o. Users had complained that the model was too *sycophantic*, showering them with praise and flattery to keep them engaged. In a blog post about the issue, OpenAI identified training on thumbs-up/down feedback from users rating their conversations as a factor that increased sycophantic behavior.[1] They noted their failure to account for users favoring more agreeable responses. In other words, OpenAI had not realized that users tend to give more positive feedback to more flattering messages. As the New York Times reported in November 2025, the failure to catch sycophantic model behavior early had serious consequences. The chatbot acted as a validating echo chamber that encouraged delusional thinking, offered suicide instructions, and caused some users to lose touch with reality for weeks.[2]

The sycophancy problem illustrates the central theme of this thesis: the importance of accounting for biases in user feedback. OpenAI's mistake was treating user feedback as a ground-truth, rather than as a flawed user signal. Just as chatbots can learn harmful patterns from users who reward flattery, search systems can learn misleading patterns from users who click top results simply because they appear first. While this thesis focuses on position bias in click data, the broader field of unbiased learning-to-rank emphasizes the importance of considering how training data for machine learning is collected. Click models, in particular, highlight this concern, as they explicitly try to capture the process generating our data. A key point of this thesis is that we must think more deeply about how users interact with systems, as applying simple bias corrections as black-box solutions does not work well. We must ask: What can users actually see on screen when providing feedback? How does the way we present information influence the type of feedback we collect? How does information displayed together depend on each other? Does positive feedback imply actual preference? Do users exhibit consistent feedback patterns, or do behaviors vary across user groups, contexts, and sessions?

While future systems may not present simple lists of search results as we considered in this thesis, as long as humans interact with machines, with our limitations, cognitive biases, and personal preferences, statistical biases in data will likely arise, requiring mechanisms to address them.

[1] https://openai.com/index/expanding-on-sycophancy/

[2] https://www.nytimes.com/2025/11/23/technology/openai-chatgpt-users-risks.html

Bibliography

[1] A. Agarwal, K. Takatsu, I. Zaitsev, and T. Joachims. A general framework for counterfactual learning-to-rank. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019. (Cited on pages 3, 12, 16, and 20.)

[2] A. Agarwal, X. Wang, C. Li, M. Bendersky, and M. Najork. Addressing trust bias for unbiased learning-to-rank. In *Proceedings of the Web Conference*, 2019. (Cited on pages 2, 28, 43, 44, and 52.)

[3] A. Agarwal, I. Zaitsev, X. Wang, C. Li, M. Najork, and T. Joachims. Estimating position bias without intrusive interventions. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019. (Cited on pages 2, 12, 34, 36, 51, 53, 54, and 63.)

[4] E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006. (Cited on page 1.)

[5] Q. Ai, K. Bi, C. Luo, J. Guo, and W. B. Croft. Unbiased learning to rank with unbiased propensity estimation. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2018. (Cited on pages 3, 18, 27, 28, 29, 31, 33, 36, 37, 38, 44, 64, 104, and 106.)

[6] Q. Ai, J. Mao, Y. Liu, and W. B. Croft. Unbiased learning to rank: Theory and practice. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018. (Cited on page 2.)

[7] Q. Ai, T. Yang, H. Wang, and J. Mao. Unbiased learning to rank: Online or offline? *ACM Transactions on Information Systems (TOIS)*, 39(2):1–29, 2021. (Cited on pages 18, 36, and 44.)

[8] M. Alaofi, P. Thomas, F. Scholer, and M. Sanderson. LLMs can be fooled into labelling a document as relevant: best café near me; this paper is perfectly relevant. In *Proceedings of the 2024 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, 2024. (Cited on page 107.)

[9] A. Amini, W. Schwarting, A. Soleimany, and D. Rus. Deep evidential regression. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, 2020. (Cited on page 106.)

[10] P. P. Analytis and P. Hager. Collaborative filtering algorithms are prone to mainstream-taste bias. In *Proceedings of the 17th ACM Conference on Recommender Systems*, 2023.

[11] A. Bai, R. Jagerman, Z. Qin, L. Yan, P. Kar, B.-R. Lin, X. Wang, M. Bendersky, and M. Najork. Regression compatible listwise objectives for calibrated ranking with binary relevance. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 2023. (Cited on pages 3 and 106.)

[12] K. Balog, D. Metzler, and Z. Qin. Rankers, judges, and assistants: Towards understanding the interplay of LLMs in information retrieval evaluation. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2025. (Cited on page 107.)

[13] J. Bekker, P. Robberechts, and J. Davis. Beyond the selected completely at random assumption for learning from positive and unlabeled data. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD*, 2019. (Cited on pages 3, 11, 12, 13, 14, 15, 31, 36, 62, 103, and 104.)

[14] G. D. Benedetto, A. Buchholz, B. London, M. Jakimov, Y. Stein, J. M. Lichtenberg, V. Bellini, and M. Ruffini. Contextual position bias estimation using a single stochastic logging policy. In *RecSys 2023 Workshop on Learning and Evaluating Recommendations with Impressions (LERI)*, 2023. (Cited on pages 54 and 63.)

[15] P. Blanchard, D. J. Higham, and N. J. Higham. Accurate computation of the log-sum-exp and softmax functions. *arXiv preprint arXiv:1909.03469*, 2019. (Cited on page 92.)

[16] C. Bonferroni. Teoria statistica delle classi e calcolo delle probabilita. *Pubblicazioni del R. Istituto Superiore di Scienze Economiche e Commerciali di Firenze*, 8:3–62, 1936. (Cited on page 20.)

[17] A. Borisov, I. Markov, M. de Rijke, and P. Serdyukov. A neural click model for web search. In *Proceedings of the 25th International Conference on World Wide Web*, 2016. (Cited on pages 12, 15, 31, 78, and 81.)

[18] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: Composable transformations of Python+NumPy programs, 2022. URL `http://github.com/google/jax`. Version 0.3.13. (Cited on pages 30, 37, 65, 79, 81, 82, 86, and 105.)

[19] S. Bruch, X. Wang, M. Bendersky, and M. Najork. An analysis of the Softmax cross entropy loss

for learning-to-rank with binary relevance. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*, 2019. (Cited on pages 31, 35, 36, and 64.)

[20] C. J. Burges. From RankNet to LambdaRank to LambdaMART: An overview. Technical Report MSR-TR-2010-82, Microsoft, 2010. (Cited on pages 12 and 18.)

[21] C. J. C. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, 2006. (Cited on pages 35 and 36.)

[22] A. Cabezas, A. Corenflos, J. Lao, and R. Louf. BlackJAX: Composable Bayesian inference in JAX. *arXiv preprint arXiv:2402.10797*, 2024. (Cited on page 82.)

[23] O. Cappé. Online expectation maximisation. In *Mixtures: Estimation and Applications*, pages 31–53. Wiley Online Library, 2011. (Cited on page 84.)

[24] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. In *Proceedings of the Learning to Rank Challenge, PMLR*, 2011. (Cited on pages 1, 12, 18, 27, 28, 33, 62, and 63.)

[25] O. Chapelle and Y. Zhang. A dynamic Bayesian network click model for web search ranking. In *Proceedings of the 18th International Conference on World Wide Web*, 2009. (Cited on pages 2, 5, 13, 30, 77, 78, 79, 80, 92, and 101.)

[26] O. Chapelle, D. Metzler, Y. Zhang, and P. Grinspan. Expected reciprocal rank for graded relevance. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, 2009. (Cited on page 18.)

[27] D. Chen, W. Chen, H. Wang, Z. Chen, and Q. Yang. Beyond ten blue links: Enabling user click modeling in federated web search. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, 2012. (Cited on page 2.)

[28] J. Chen, J. Mao, Y. Liu, M. Zhang, and S. Ma. A context-aware click model for web search. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020. (Cited on pages 13, 31, 78, 81, and 106.)

[29] J. Chen, H. Li, W. Su, Q. Ai, and Y. Liu. THUIR at WSDM Cup 2023 Task 1: Unbiased learning to rank. In *Proceedings of the 16th ACM International Conference on Web Search and Data Mining*, 2023. (Cited on pages 31 and 33.)

[30] M. Chen, C. Liu, Z. Liu, Z. Li, and J. Sun. Identifiability matters: Revealing the hidden recoverable condition in unbiased learning to rank. In *Proceedings of the 41st International Conference on Machine Learning*, 2024. (Cited on pages 4, 43, 51, 53, 55, 56, 57, 58, 60, and 104.)

[31] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, 13(4):359–394, 1999. (Cited on page 33.)

[32] W. Chen, D. Wang, Y. Zhang, Z. Chen, A. Singla, and Q. Yang. A noise-aware click model for web search. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, 2012. (Cited on page 1.)

[33] X. Chen, X. Li, K. Wei, B. Hu, L. Jiang, Z. Huang, and Z. Kang. Multi-feature integration for perception-dependent examination-bias estimation. In *Proceedings of the 16th ACM International Conference on Web Search and Data Mining*, 2023. (Cited on pages 31, 37, 44, 51, 52, 54, 59, 73, and 93.)

[34] Y. Chen and T. W. Yan. Position-normalized click prediction in search advertising. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2012. (Cited on page 78.)

[35] W. Chu, S. Li, C. Chen, L. Xu, H. Cui, and K. Liu. A general framework for debiasing in CTR prediction. *arXiv preprint arXiv:2112.02767*, 2021. (Cited on pages 12 and 15.)

[36] A. Chuklin, P. Serdyukov, and M. de Rijke. Click model-based information retrieval metrics. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2013. (Cited on page 78.)

[37] A. Chuklin, I. Markov, and M. de Rijke. *Click Models for Web Search*. Morgan & Claypool, 2015. (Cited on pages 2, 11, 12, 13, 30, 53, 77, 78, 79, 80, 81, 82, 89, 90, 93, 99, 100, 103, and 105.)

[38] P. Covington, J. Adams, and E. Sargin. Deep neural networks for YouTube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, 2016. (Cited on page 12.)

[39] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*, 2008. (Cited on pages 2, 11, 12, 13, 14, 28, 30, 51, 52, 53, 55, 57, 77, 78, 80, and 82.)

[40] A. D'Amour, P. Ding, A. Feller, L. Lei, and J. Sekhon. Overlap in observational studies with high-dimensional covariates. *Journal of Econometrics*, 221(2):644–654, 2021. (Cited on pages 59, 67, 71, and 105.)

[41] D. Dato, C. Lucchese, F. M. Nardini, S. Orlando, R. Perego, N. Tonellotto, and R. Venturini. Fast ranking with additive ensembles of oblivious and non-oblivious regression trees. *ACM Transactions on Information Systems (TOIS)*, 35(2):1–31, 2016. (Cited on pages 1, 18, 27, 28, 33, and 63.)

[42] D. Dato, S. MacAvaney, F. M. Nardini, R. Perego, and N. Tonellotto. The Istella22 dataset: Bridging traditional and neural learning to rank evaluation. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022. (Cited on pages 28, 33, 62, and 63.)

[43] M. de Haan and P. Hager. Understanding the effects of the Baidu-ULTR logging policy on two-tower models. In *The CONSEQUENCES'24 workshop, co-located with ACM RecSys'24*, 2024.

[44] S. de Leon-Martinez, J. Kang, R. Moro, M. de Rijke, B. Kveton, H. Oosterhuis, and M. Bielikova. Recgaze: The first eye tracking and user interaction dataset for carousel interfaces. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2025. (Cited on page 106.)

[45] S. de Leon-Martinez, R. Moro, B. Kveton, and M. Bielikova. Riding the carousel: The first extensive eye tracking analysis of browsing behavior in carousel recommenders. *arXiv preprint arXiv:2507.10135*, 2025. (Cited on page 106.)

[46] DeepMind, I. Babuschkin, K. Baumli, A. Bell, S. Bhupatiraju, J. Bruce, P. Buchlovsky, D. Budden, T. Cai, A. Clark, I. Danihelka, A. Dedieu, C. Fantacci, J. Godwin, C. Jones, R. Hemsley, T. Hennigan, M. Hessel, S. Hou, S. Kapturowski, T. Keck, I. Kemaev, M. King, M. Kunesch, L. Martens, H. Merzic, V. Mikulik, T. Norman, G. Papamakarios, J. Quan, R. Ring, F. Ruiz, A. Sanchez, L. Sartran, R. Schneider, E. Sezener, S. Spencer, S. Srinivasan, M. Stanojević, W. Stokowiec, L. Wang, G. Zhou, and F. Viola. The DeepMind JAX ecosystem, 2020. (Cited on pages 80, 82, and 105.)

[47] R. Deffayet, P. Hager, J.-M. Renders, and M. de Rijke. An offline metric for the debiasedness of click models. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2023. (Cited on pages 43, 53, 54, 62, 63, 67, 75, and 107.)

[48] R. Deffayet, J.-M. Renders, and M. de Rijke. Evaluating the robustness of click models to policy distributional shift. *ACM Transactions on Information Systems (TOIS)*, 41(4), 2023. (Cited on pages 3, 27, 43, 54, 63, 75, 78, 81, and 107.)

[49] R. Deffayet, T. Thonet, D. Hwang, V. Lehoux, J.-M. Renders, and M. de Rijke. SARDINE: Simulator for automated recommendation in dynamic and interactive environments. *ACM Transactions on Recommender Systems (TORS)*, 2(3), 2024. (Cited on page 78.)

[50] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977. (Cited on pages 36, 82, 84, and 105.)

[51] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2019. (Cited on page 37.)

[52] F. Diaz, R. White, G. Buscher, and D. Liebling. Robust models of mouse movement on dynamic web search results pages. In *Proceedings of the 22nd ACM International Conference on Information and Knowledge Management*, 2013. (Cited on page 13.)

[53] L. Dietz, O. Zendel, P. Bailey, C. L. A. Clarke, E. Cotterill, J. Dalton, F. Hasibi, M. Sanderson, and N. Craswell. Principles and guidelines for the use of LLM judges. In *Proceedings of the 2025 International ACM SIGIR Conference on Innovative Concepts and Theories in Information Retrieval*, 2025. (Cited on page 107.)

[54] T. Draws, N. Tintarev, U. Gadiraju, A. Bozzon, and B. Timmermans. This is not what we ordered: Exploring why biased search result rankings affect user attitudes on debated topics. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021. (Cited on page 107.)

[55] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12:2121–2159, 2011. (Cited on page 84.)

[56] G. E. Dupret and B. Piwowarski. A user browsing model to predict search engine click data from past observations. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2008. (Cited on pages 5, 12, 13, 30, 77, 78, 79, 80, 87, and 90.)

[57] F. Eberhardt. Introduction to the foundations of causal discovery. *International Journal of Data Science and Analytics*, 3(2):81–91, 2017. (Cited on page 107.)

[58] R. Epstein and R. E. Robertson. The search engine manipulation effect (seme) and its possible impact on the outcomes of elections. *Proceedings of the National Academy of Sciences*, 112(33):4512–4521, 2015. (Cited on page 107.)

[59] G. Faggioli, L. Dietz, C. L. A. Clarke, G. Demartini, M. Hagen, C. Hauff, N. Kando, E. Kanoulas, M. Potthast, B. Stein, and H. Wachsmuth. Perspectives on large language models for relevance judgment. In *Proceedings of the 2023 ACM SIGIR International Conference on Theory of Information Retrieval*, 2023. (Cited on page 107.)

[60] Z. Fang, A. Agarwal, and T. Joachims. Intervention harvesting for context-dependent examination-bias estimation. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019. (Cited on pages 2, 53, 54, and 63.)

[61] T. Formal, B. Piwowarski, and S. Clinchant. SPLADE: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021. (Cited on page 27.)

[62] S. Fox, K. Karnawat, M. Mydland, S. Dumais, and T. White. Evaluating implicit measures to improve web search. *ACM Transactions on Information Systems (TOIS)*, 23(2):147–168, 2005. (Cited on pages 1 and 106.)

[63] M. Fröbe, A. Parry, F. Schlatt, S. MacAvaney, B. Stein, M. Potthast, and M. Hagen. Large language model relevance assessors agree with one another more than with human assessors. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2025. (Cited on page 107.)

[64] V. Godbole, G. E. Dahl, J. Gilmer, C. J. Shallue, and Z. Nado. Deep learning tuning playbook, 2023. URL `http://github.com/google-research/tuning_playbook`. Version 1.0. (Cited on page 38.)

[65] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)*, 23(1):5–48, 1991. (Cited on pages 91 and 92.)

[66] C. A. Gomez-Uribe and N. Hunt. The Netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4), 2016. (Cited on page 12.)

[67] L. A. Granka, T. Joachims, and G. Gay. Eye-tracking analysis of user behavior in WWW search. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2004. (Cited on pages 2 and 11.)

[68] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. Smola. A kernel two-sample test. *The Journal of Machine Learning Research (JMLR)*, 13:723–773, 2012. (Cited on page 70.)

[69] A. Grotov, A. Chuklin, I. Markov, L. Stout, F. Xumara, and M. de Rijke. A comparative study of click models for web search. In *Proceedings of the 6th International Conference on Experimental IR Meets Multilinguality, Multimodality, and Interaction*, 2015. (Cited on page 89.)

[70] F. Guo, C. Liu, A. Kannan, T. Minka, M. Taylor, Y.-M. Wang, and C. Faloutsos. Click chain model in web search. In *Proceedings of the 18th International Conference on World Wide Web*, 2009. (Cited on pages 12, 13, 77, 78, and 80.)

[71] F. Guo, C. Liu, and Y. M. Wang. Efficient multiple-click models in web search. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, 2009. (Cited on pages 80 and 95.)

[72] H. Guo, J. Yu, Q. Liu, R. Tang, and Y. Zhang. PAL: A position-bias aware learning framework for CTR prediction in live recommender systems. In *Proceedings of the 13th ACM Conference on Recommender Systems*, 2019. (Cited on pages 2, 4, 11, 12, 13, 15, 25, 31, 48, 51, 52, 54, 55, 78, 79, 81, 88, and 104.)

[73] R. Guo, X. Zhao, A. Henderson, L. Hong, and H. Liu. Debiasing grid-based product search in e-commerce. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020. (Cited on page 106.)

[74] S. Gupta, P. Hager, J. Huang, A. Vardasbi, and H. Oosterhuis. Recent advances in the foundations and applications of unbiased learning to rank. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2023.

[75] S. Gupta, H. Oosterhuis, and M. de Rijke. Safe deployment for counterfactual learning to rank with exposure-based risk minimization. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2023. (Cited on pages 3, 27, and 43.)

[76] S. Gupta, P. Hager, J. Huang, A. Vardasbi, and H. Oosterhuis. Unbiased learning to rank: On recent advances and practical applications. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, 2024. (Cited on pages 2, 28, and 52.)

[77] H. Habib, M. Li, E. Young, and L. Cranor. "okay, whatever": An evaluation of cookie consent interfaces. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022. (Cited on page 107.)

[78] P. Hager, M. de Rijke, and O. Zoeter. Are neural click models pointwise IPS rankers? In *The CONSEQUENCES'22 workshop, co-located with ACM RecSys'22*, CONSEQUENCES'22, 2022.

[79] P. Hager, M. de Rijke, and O. Zoeter. Contrasting neural click models and pointwise IPS rankers. In *Advances in Information Retrieval: 45th European Conference on Information Retrieval*, 2023. (Cited on pages 54, 62, and 103.)

[80] P. Hager, R. Deffayet, J.-M. Renders, O. Zoeter, and M. de Rijke. Unbiased learning to rank meets reality: Lessons from Baidu's large-scale search dataset. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024. (Cited on pages 79, 93, 96, 97, 98, 104, and 106.)

[81] P. Hager, M. de Rijke, and O. Zoeter. Unidentified and confounded? Understanding two-tower models for unbiased learning to rank (extended abstract). In *The CONSEQUENCES'25 workshop, co-located with ACM RecSys'25*, 2025.

[82] P. Hager, O. Zoeter, and M. de Rijke. CLAX: Fast and flexible neural click models in JAX. *arXiv preprint arXiv:2511.03620*, 2025. (Cited on pages 105 and 106.)

[83] P. Hager, O. Zoeter, and M. de Rijke. Unidentified and confounded? Understanding two-tower models for unbiased learning to rank. In *Proceedings of the 2025 International ACM SIGIR Conference on Innovative Concepts and Theories in Information Retrieval*, 2025. (Cited on page 104.)

[84] M. Haldar, M. Abdool, P. Ramanathan, T. Xu, S. Yang, H. Duan, Q. Zhang, N. Barrow-Williams, B. C. Turnbull, B. M. Collins, and T. Legrand. Applying deep learning to Airbnb search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019. (Cited on page 1.)

[85] M. Haldar, P. Ramanathan, T. Sax, M. Abdool, L. Zhang, A. Mansawala, S. Yang, B. Turnbull, and J. Liao. Improving deep learning for Airbnb search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020. (Cited on pages 2, 12, 13, 15, 25, 48, 51, 52, 54, 55, 78, 79, 81, and 104.)

[86] J. Heek, A. Levskaya, A. Oliver, M. Ritter, B. Rondepierre, A. Steiner, and M. van Zee. Flax: A neural network library and ecosystem for JAX, 2024. URL `http://github.com/google/flax`. Version 0.10.5. (Cited on pages 65, 79, 81, and 82.)

[87] T. Hennigan, T. Cai, T. Norman, L. Martens, and I. Babuschkin. Haiku: Sonnet for JAX, 2020. URL `http://github.com/deepmind/dm-haiku`. Version 0.0.14. (Cited on page 82.)

[88] K. Hofmann, A. Schuth, S. Whiteson, and M. de Rijke. Reusing historical interaction data for faster online learning to rank for IR. In *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, 2013. (Cited on page 17.)

[89] J. Hu, H.-J. Zeng, H. Li, C. Niu, and Z. Chen. Demographic prediction based on user's browsing behavior. In *Proceedings of the 16th International Conference on World Wide Web*, 2007. (Cited on page 107.)

[90] Z. Hu, Y. Wang, Q. Peng, and H. Li. Unbiased LambdaMART: An unbiased pairwise learning-to-rank algorithm. In *Proceedings of the Web Conference*, 2019. (Cited on pages 12, 13, 18, 29, 31, 36, 44, and 104.)

[91] J. Huang, H. Oosterhuis, M. de Rijke, and H. van Hoof. Keeping dataset biases out of the simulation: A debiased simulator for reinforcement learning based recommender systems. In *Proceedings of the 14th ACM Conference on Recommender Systems*, 2020. (Cited on page 78.)

[92] L. Hurwicz. Generalization of the concept of identification. *Statistical Inference in Dynamic Economic Models*, 10:245–57, 1950. (Cited on page 55.)

[93] R. Jagerman, H. Oosterhuis, and M. de Rijke. To model or to intervene: A comparison of counterfactual and online learning to rank from user interactions. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2019. (Cited on pages 17 and 64.)

[94] R. Jagerman, X. Wang, H. Zhuang, Z. Qin, M. Bendersky, and M. Najork. Rax: Composable learning-to-rank using JAX. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022. (Cited on pages 30, 37, 65, 82, 86, and 90.)

[95] O. Jeunen. A probabilistic position bias model for short-video recommendation feeds. In *Proceedings of the 17th ACM Conference on Recommender Systems*, 2023. (Cited on page 106.)

[96] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002. (Cited on page 1.)

[97] T. Joachims, L. Granka, B. Pan, H. Hembrooke, and G. Gay. Accurately interpreting clickthrough data as implicit feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference*

on Research and Development in Information Retrieval, 2005. (Cited on pages 2, 11, 12, 28, 52, 55, and 106.)

[98] T. Joachims, L. Granka, B. Pan, H. Hembrooke, F. Radlinski, and G. Gay. Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search. *ACM Transactions on Information Systems (TOIS)*, 25(2):7–34, 2007. (Cited on page 2.)

[99] T. Joachims, A. Swaminathan, and T. Schnabel. Unbiased learning-to-rank with biased feedback. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, 2017. (Cited on pages 2, 3, 11, 12, 13, 15, 16, 17, 20, 22, 27, 28, 29, 31, 33, 36, 44, 52, 53, 54, 57, 62, 64, 103, and 104.)

[100] W. Kahan. IEEE standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776):11, 1996. (Cited on page 91.)

[101] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. LightGBM: A highly efficient gradient boosting decision tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017. (Cited on page 18.)

[102] P. Khandel, I. Markov, A. Yates, and A.-L. Varbanescu. ParClick: A scalable algorithm for EM-based click models. In *Proceedings of the ACM Web Conference*, 2022. (Cited on pages 5, 77, 78, and 81.)

[103] E. Kharitonov. Federated online learning to rank with evolution strategies. In *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, 2019. (Cited on page 107.)

[104] O. Khattab and M. Zaharia. ColBERT: Efficient and effective passage search via contextualized late interaction over BERT. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020. (Cited on page 27.)

[105] K. Khrylchenko and A. Fritzler. Personalized transformer-based ranking for e-commerce at Yandex. *arXiv preprint arXiv:2310.03481*, 2023. (Cited on pages 2, 48, 51, 52, 54, and 104.)

[106] P. Kidger. *On Neural Differential Equations*. PhD thesis, University of Oxford, 2021. (Cited on page 82.)

[107] E. Kim, S. M. Choi, S. Kim, and Y.-H. Yeh. Factors affecting advertising avoidance on online video sites. *The Journal of Advertising and Promotion Research*, 2(1):87–121, 2013. (Cited on page 1.)

[108] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. (Cited on page 84.)

[109] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009. (Cited on pages 56, 77, 84, and 105.)

[110] S. Kolouri, K. Nadjahi, U. Simsekli, R. Badeau, and G. Rohde. Generalized sliced Wasserstein distances. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019. (Cited on page 70.)

[111] T. C. Koopmans and O. Reiersol. The identification of structural characteristics. *The Annals of Mathematical Statistics*, 21(2):165–181, 1950. (Cited on page 55.)

[112] A. Lewbel. The identification zoo: Meanings of identification in econometrics. *Journal of Economic Literature*, 57(4):835–903, 2019. (Cited on pages 55, 57, 58, and 70.)

[113] H. Li, J. Chen, W. Su, Q. Ai, and Y. Liu. Towards better web search performance: Pre-training, fine-tuning and learning to rank. In *Proceedings of the 16th ACM International Conference on Web Search and Data Mining*, 2023. (Cited on page 31.)

[114] J. Li, S. Huffman, and A. Tokuda. Good abandonment in mobile and PC internet search. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2009. (Cited on page 2.)

[115] S. Li, Y. Abbasi-Yadkori, B. Kveton, S. Muthukrishnan, V. Vinay, and Z. Wen. Offline evaluation of ranking policies with click models. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2018. (Cited on page 63.)

[116] X. Li, X. Chen, K. Wei, B. Hu, L. Jiang, Z. Huang, and Z. Kang. Pretraining de-biased language model with large-scale click logs for document ranking. In *Proceedings of the 16th ACM International Conference on Web Search and Data Mining*, 2023. (Cited on pages 31, 32, and 93.)

[117] J. Lin, W. Liu, X. Dai, W. Zhang, S. Li, R. Tang, X. He, J. Hao, and Y. Yu. A graph-enhanced click model for web search. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2021. (Cited on pages 13, 78, and 81.)

[118] J. Lin, R. Nogueira, and A. Yates. Pretrained transformers for text ranking: BERT and beyond. *arXiv preprint arXiv:2010.06467*, 2021. (Cited on pages 27 and 104.)

[119] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009. (Cited on pages 1, 2, and 12.)

[120] D. Lopez-Paz and M. Oquab. Revisiting classifier two-sample tests. In *The International Conference*

on Learning Representations, 2017. (Cited on page 70.)

[121] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2019. (Cited on pages 38, 65, 84, and 93.)

[122] D. Luo, L. Zou, Q. Ai, Z. Chen, C. Li, D. Yin, and B. D. Davison. Unbiased learning-to-rank needs unconfounded propensity estimation. In *The 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024. (Cited on pages 4, 51, 53, 54, 60, 63, 75, and 104.)

[123] M. Mächler. Accurately computing $\log(1 - \exp(-|a|))$ assessed by the Rmpfr package. *The Comprehensive R Archive Network*, pages 1–9, 2012. (Cited on page 92.)

[124] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. In *The International Conference on Learning Representations*, 2017. (Cited on page 64.)

[125] J. Mao, C. Luo, M. Zhang, and S. Ma. Constructing click models for mobile search. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2018. (Cited on page 2.)

[126] J. Mao, Z. Chu, Y. Liu, M. Zhang, and S. Ma. Investigating the reliability of click models. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*, 2019. (Cited on page 106.)

[127] I. Markov, A. Borisov, and M. de Rijke. Online expectation-maximization for click models. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, 2017. (Cited on pages 5, 77, and 78.)

[128] A. Mathur, G. Acar, M. J. Friedman, E. Lucherini, J. Mayer, M. Chetty, and A. Narayanan. Dark patterns at scale: Findings from a crawl of 11k shopping websites. *Proceedings of the ACM on Human-Computer Interaction*, 2019. (Cited on page 107.)

[129] H. B. McMahan, G. Holt, D. Sculley, M. Young, D. Ebner, J. Grady, L. Nie, T. Phillips, E. Davydov, D. Golovin, S. Chikkerur, D. Liu, M. Wattenberg, A. M. Hrafnkelsson, T. Boulos, and J. Kubica. Ad click prediction: A view from the trenches. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013. (Cited on page 78.)

[130] L. Minto, M. Haller, B. Livshits, and H. Haddadi. Stronger privacy for federated collaborative filtering with implicit feedback. In *Proceedings of the 15th ACM Conference on Recommender Systems*, 2021. (Cited on page 107.)

[131] R. M. Neal and G. E. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In *Learning in graphical models*, pages 355–368. Springer, 1998. (Cited on page 84.)

[132] Z. Niu, J. Mao, Q. Ai, L. Zou, S. Wang, and D. Yin. Overview of the NTCIR-17 unbiased learning to rank evaluation 2 (ULTRE-2) task. In *The 17th Round of NII Testbeds and Community for Information Access Research*, 2023. (Cited on pages 29, 31, 32, 33, and 39.)

[133] R. Nogueira, W. Yang, K. Cho, and J. Lin. Multi-stage document ranking with BERT. *arXiv preprint arXiv:1910.14424*, 2019. (Cited on pages 27, 29, 31, 37, 41, and 104.)

[134] D. W. Oard and J. Kim. Implicit feedback for recommender systems. In *Proceedings of the AAAI workshop on recommender systems*, volume 83, 1998. (Cited on page 1.)

[135] H. Oosterhuis. Reaching the end of unbiasedness: Uncovering implicit limitations of click-based learning to rank. In *Proceedings of the 2022 ACM SIGIR International Conference on Theory of Information Retrieval*, 2022. (Cited on pages 3, 4, 12, 14, 16, 36, 43, 55, and 57.)

[136] H. Oosterhuis and M. de Rijke. Differentiable unbiased online learning to rank. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018. (Cited on page 17.)

[137] H. Oosterhuis and M. de Rijke. Policy-aware unbiased learning to rank for top-k rankings. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020. (Cited on pages 1, 2, 3, 12, 13, 16, 20, 27, 54, 62, 63, and 64.)

[138] H. Oosterhuis and M. de Rijke. Unifying online and counterfactual learning to rank: A novel counterfactual estimator that effectively utilizes online interventions. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021. (Cited on pages 12, 13, 17, 28, 31, 37, and 38.)

[139] H. Oosterhuis, R. Jagerman, and M. de Rijke. Unbiased learning to rank: Counterfactual and online approaches. In *Proceedings of the Web Conference*, 2020. (Cited on pages 1 and 2.)

[140] Z. Ovaisi, R. Ahsan, Y. Zhang, K. Vasilaky, and E. Zheleva. Correcting for selection bias in learning-to-rank systems. In *Proceedings of the Web Conference*, 2020. (Cited on pages 3, 17, and 27.)

[141] R. K. Pasumarthi, S. Bruch, X. Wang, C. Li, M. Bendersky, M. Najork, J. Pfeifer, N. Golbandi, R. Anil, and S. Wolf. TF-Ranking: Scalable TensorFlow library for learning-to-rank. In *Proceedings of the*

25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2019. (Cited on page 36.)

[142] B. L. Pereira, A. Ueda, G. Penha, R. L. T. Santos, and N. Ziviani. Online learning to rank for sequential music recommendation. In *Proceedings of the 13th ACM Conference on Recommender Systems*, 2019. (Cited on page 1.)

[143] R. Pradeep, R. Nogueira, and J. Lin. The expando-mono-duo design pattern for text ranking with pretrained sequence-to-sequence models. *arXiv preprint arXiv:2101.05667*, 2021. (Cited on page 29.)

[144] N. Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1): 145–151, 1999. (Cited on page 84.)

[145] T. Qin and T.-Y. Liu. Introducing LETOR 4.0 datasets. *arXiv preprint arXiv:1306.2597*, 2013. (Cited on pages 1, 12, 18, 27, 28, 33, 62, and 63.)

[146] Z. Qin, L. Yan, H. Zhuang, Y. Tay, R. K. Pasumarthi, X. Wang, M. Bendersky, and M. Najork. Are neural rankers still outperformed by gradient boosted decision trees? In *The International Conference on Learning Representations*, 2021. (Cited on pages 18, 38, and 63.)

[147] J. Rader, T. Lyons, and P. Kidger. Lineax: Unified linear solves and linear least-squares in JAX and Equinox. In *AI for science workshop at Neural Information Processing Systems*, 2023. (Cited on page 82.)

[148] F. Radlinski and T. Joachims. Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. In *Proceedings of the 21st National Conference on Artificial Intelligence*, 2006. (Cited on pages 57 and 64.)

[149] H. A. Rahmani, N. Craswell, E. Yilmaz, B. Mitra, and D. Campos. Synthetic test collections for retrieval evaluation. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024. (Cited on page 107.)

[150] H. A. Rahmani, C. Siro, M. Aliannejadi, N. Craswell, C. L. A. Clarke, G. Faggioli, B. Mitra, P. Thomas, and E. Yilmaz. Judging the judges: A collection of LLM-generated relevance judgements. *arXiv preprint arXiv:2502.13908*, 2025. (Cited on page 107.)

[151] O. Ramirez Milian and H. Oosterhuis. Epistemic click models through evidential deep learning. In *The CONSEQUENCES'25 workshop, co-located with ACM RecSys'25*, 2025. (Cited on page 106.)

[152] A. Raue, C. Kreutz, T. Maiwald, J. Bachmann, M. Schilling, U. Klingmüller, and J. Timmer. Structural and practical identifiability analysis of partially observed dynamical models by exploiting the profile likelihood. *Bioinformatics*, 25(15):1923–1929, 2009. (Cited on page 70.)

[153] M. Richardson, E. Dominowska, and R. Ragno. Predicting clicks: Estimating the click-through rate for new ads. In *Proceedings of the 16th International Conference on World Wide Web*, 2007. (Cited on pages 2, 11, 12, 13, 14, 28, 30, 55, 57, 77, 78, 80, 82, and 103.)

[154] S. Robertson. Understanding inverse document frequency: On theoretical arguments for IDF. *Journal of documentation*, 60(5):503–520, 2004. (Cited on pages 33 and 37.)

[155] S. E. Robertson, S. Walker, S. Jones, M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proceedings of The Third Text REtrieval Conference, TREC*, 1994. (Cited on pages 33, 37, 42, and 104.)

[156] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2017. (Cited on page 84.)

[157] Y. Saito, S. Yaginuma, Y. Nishino, H. Sakata, and K. Nakata. Unbiased recommender learning from missing-not-at-random implicit feedback. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, 2020. (Cited on pages 3, 11, 12, 13, 14, 15, 16, 31, 36, 44, 62, and 103.)

[158] R. Salakhutdinov, S. T. Roweis, and Z. Ghahramani. Optimization with EM and expectation-conjugate-gradient. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, 2003. (Cited on pages 84, 85, and 102.)

[159] M. Sanderson. Test collection based evaluation of information retrieval. *Foundations and Trends in Information Retrieval*, 4(4):247–375, 2010. (Cited on pages 12 and 28.)

[160] F. Sarvi, A. Vardasbi, M. Aliannejadi, S. Schelter, and M. de Rijke. On the impact of outlier bias on user clicks. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2023. (Cited on pages 2, 43, and 106.)

[161] K. Scaman and A. Virmaux. Lipschitz regularity of deep neural networks: analysis and efficient estimation. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018. (Cited on pages 59 and 105.)

[162] P. Serdyukov, N. Craswell, and G. Dupret. WSCD 2012: Workshop on web search click data 2012. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*, 2012. (Cited on page 93.)

[163] H.-J. M. Shi, D. Mudigere, M. Naumov, and J. Yang. Compositional embeddings using complementary partitions for memory-efficient recommendation systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020. (Cited on pages 79, 88, and 105.)

[164] D. Sorokina and E. Cantu-Paz. Amazon search: The joy of ranking products. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2016. (Cited on page 12.)

[165] Student. The probable error of a mean. *Biometrika*, pages 1–25, 1908. (Cited on page 20.)

[166] X. Sun, L. Yu, Y. Wang, K. Bi, and J. Guo. Ensemble ranking model with multiple pretraining strategies for web search. In *Proceedings of the Sixteen ACM International Conference on Web Search and Data Mining*, 2023. (Cited on pages 31, 32, and 42.)

[167] A. Swaminathan and T. Joachims. The self-normalized estimator for counterfactual learning. In *Proceedings of the 29th International Conference on Neural Information Processing Systems*, 2015. (Cited on page 17.)

[168] R. Takehi, E. M. Voorhees, T. Sakai, and I. Soboroff. LLM-assisted relevance assessments: When should we ask llms for help? In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2025. (Cited on pages 107 and 108.)

[169] N. Thakur, R. Pradeep, S. Upadhyay, D. Campos, N. Craswell, I. Soboroff, H. T. Dang, and J. Lin. Assessing support for the trec 2024 rag track: A large-scale comparative study of LLM and human evaluations. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2025. (Cited on page 107.)

[170] B. Thiesson, C. Meek, and D. Heckerman. Accelerating EM for large databases. *Machine Learning*, 45(3):279–299, 2001. (Cited on page 84.)

[171] S. Thijssen, P. Khandel, A. Yates, and A.-L. Varbanescu. MassiveClicks: A massively-parallel framework for efficient click models training. In *European Conference on Parallel Processing*, 2023. (Cited on pages 5, 77, 78, and 81.)

[172] P. Thomas, S. Spielman, N. Craswell, and B. Mitra. Large language models can accurately predict searcher preferences. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024. (Cited on page 107.)

[173] A. Tran, T. Yang, and Q. Ai. ULTRA: An unbiased learning to rank algorithm toolbox. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021. (Cited on pages 29 and 44.)

[174] C. Utz, M. Degeling, S. Fahl, F. Schaub, and T. Holz. (un)informed consent: Studying gdpr consent notices in the field. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019. (Cited on page 107.)

[175] A. Vardasbi, M. de Rijke, and I. Markov. Cascade model-based propensity estimation for counterfactual learning to rank. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020. (Cited on pages 2, 11, 13, 20, 28, 38, 43, and 44.)

[176] A. Vardasbi, H. Oosterhuis, and M. de Rijke. When inverse propensity scoring does not work: Affine corrections for unbiased learning to rank. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020. (Cited on pages 2, 3, 11, 12, 16, 17, 18, 20, 27, 28, 31, 37, 52, 54, and 78.)

[177] A. Vardasbi, M. de Rijke, and I. Markov. Mixture-based correction for position and trust bias in counterfactual learning to rank. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021. (Cited on pages 12, 17, and 18.)

[178] J. Vonásek, M. Straka, R. Krč, L. Lasonová, E. Egorova, J. Straková, and J. Náplava. CWRCzech: 100M query-document czech click dataset and its application to web relevance ranking. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2024. (Cited on page 106.)

[179] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1–2):1–305, 2008. (Cited on page 106.)

[180] C. Wang, Y. Liu, M. Zhang, S. Ma, M. Zheng, J. Qian, and K. Zhang. Incorporating vertical results into search click models. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2013. (Cited on page 106.)

[181] C. Wang, Y. Liu, M. Wang, K. Zhou, J.-y. Nie, and S. Ma. Incorporating non-sequential behavior into click models. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2015. (Cited on page 13.)

[182] R. Wang, R. Shivanna, D. Cheng, S. Jain, D. Lin, L. Hong, and E. Chi. DCN V2: Improved deep &

cross network and practical lessons for web-scale learning to rank systems. In *Proceedings of the Web Conference*, 2021. (Cited on pages 79, 88, and 105.)

[183] X. Wang, M. Bendersky, D. Metzler, and M. Najork. Learning to rank with selection bias in personal search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2016. (Cited on pages 1, 2, 3, 12, 13, 28, 29, 31, and 52.)

[184] X. Wang, N. Golbandi, M. Bendersky, D. Metzler, and M. Najork. Position bias estimation for unbiased learning to rank in personal search. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, 2018. (Cited on pages 2, 11, 12, 13, 14, 25, 29, 31, 34, 36, 78, 85, 103, and 104.)

[185] Y.-X. Wang, A. Agarwal, and M. Dudík. Optimal and adaptive off-policy evaluation in contextual bandits. In *Proceedings of the 34th International Conference on Machine Learning*, 2017. (Cited on page 17.)

[186] C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989. (Cited on page 64.)

[187] I. Weber and C. Castillo. The demographics of web search. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010. (Cited on page 107.)

[188] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, 2009. (Cited on pages 79, 88, and 105.)

[189] H. Wen, L. Yang, and D. Estrin. Leveraging post-click feedback for content recommendations. In *Proceedings of the 13th ACM Conference on Recommender Systems*, 2019. (Cited on page 1.)

[190] R. Wilms, E. Mäthner, L. Winnen, and R. Lanwehr. Omitted variable bias: A threat to estimating causal relationships. *Methods in Psychology*, 5, 2021. (Cited on page 62.)

[191] C. F. J. Wu. On the convergence properties of the EM algorithm. *The Annals of Statistics*, 11(1): 95–103, 1983. (Cited on page 84.)

[192] X. Wu, H. Chen, J. Zhao, L. He, D. Yin, and Y. Chang. Unbiased learning to rank in feeds recommendation. In *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021. (Cited on pages 52, 54, and 106.)

[193] X. Xie, Y. Liu, X. Wang, M. Wang, Z. Wu, Y. Wu, M. Zhang, and S. Ma. Investigating examination behavior of image search users. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2017. (Cited on page 13.)

[194] X. Xie, J. Mao, M. de Rijke, R. Zhang, M. Zhang, and S. Ma. Constructing an interaction behavior model for web image search. In *Proceedings of the 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2018. (Cited on pages 13 and 106.)

[195] L. Yan, Z. Qin, H. Zhuang, X. Wang, M. Bendersky, and M. Najork. Revisiting two-tower models for unbiased learning to rank. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022. (Cited on pages 2, 3, 4, 11, 12, 13, 15, 25, 31, 35, 36, 38, 51, 52, 53, 54, 55, 59, 73, 78, 79, 81, 88, 89, 97, 98, 103, 104, 105, and 106.)

[196] L. Yu, K. Bi, J. Guo, and X. Cheng. CIR at the NTCIR-17 ULTRE-2 task. In *The 17th Round of NII Testbeds and Community for Information Access Research (NTCIR)*, 2023. (Cited on pages 31 and 32.)

[197] L. Yu, Y. Wang, X. Sun, K. Bi, and J. Guo. Feature-enhanced network with hybrid debiasing strategies for unbiased learning to rank. In *Proceedings of the Sixteen ACM International Conference on Web Search and Data Mining*, 2023. (Cited on page 31.)

[198] Y. Yue, R. Patel, and H. Roehrig. Beyond position bias: Examining result attractiveness as a source of presentation bias in clickthrough data. In *Proceedings of the 19th International Conference on World Wide Web*, 2010. (Cited on page 1.)

[199] Y. Zhang, D. Wang, G. Wang, W. Chen, Z. Zhang, B. Hu, and L. Zhang. Learning click models via probit bayesian inference. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, 2010. (Cited on page 106.)

[200] Y. Zhang, L. Yan, Z. Qin, H. Zhuang, J. Shen, X. Wang, M. Bendersky, and M. Najork. Towards disentangling relevance and bias in unbiased learning to rank. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023. (Cited on pages 4, 31, 36, 51, 52, 53, 54, 60, 62, 64, 67, 71, 74, 104, and 105.)

[201] Y. Zhao, Z. Niu, F. Wang, J. Mao, Q. Ai, Y. Tao, J. Zhang, and Y. Liu. Overview of the NTCIR-16 unbiased learning to rank evaluation (ULTRE) task. In *The 16th Round of NII Testbeds and Community for Information Access Research (NTCIR)*, 2022. (Cited on pages 31, 38, and 39.)

[202] Z. Zhao, L. Hong, L. Wei, J. Chen, A. Nath, S. Andrews, A. Kumthekar, M. Sathiamoorthy, X. Yi, and

E. Chi. Recommending what video to watch next: A multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems*, 2019. (Cited on pages 2, 4, 12, 13, 15, 25, 31, 51, 52, 54, 55, 78, 88, and 104.)

[203] F. Zhong, D. Wang, G. Wang, W. Chen, Y. Zhang, Z. Chen, and H. Wang. Incorporating post-click behaviors into a click model. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2010. (Cited on pages 2 and 106.)

[204] Z. Zhu, R. Qin, J. Huang, X. Dai, Y. Yu, Y. Yu, and W. Zhang. Understanding or manipulation: Rethinking online performance gains of modern recommender systems. *ACM Transactions on Information Systems (TOIS)*, 42(4), 2024. (Cited on page 107.)

[205] Z. A. Zhu, W. Chen, T. Minka, C. Zhu, and Z. Chen. A novel click model and its applications to online advertising. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, 2010. (Cited on pages 2 and 78.)

[206] H. Zhuang, Z. Qin, X. Wang, M. Bendersky, X. Qian, P. Hu, and D. C. Chen. Cross-positional attention for debiasing clicks. In *Proceedings of the Web Conference*, 2021. (Cited on pages 13, 15, 28, 31, 43, 52, 54, 78, and 81.)

[207] H. Zhuang, Z. Qin, R. Jagerman, K. Hui, J. Ma, J. Lu, J. Ni, X. Wang, and M. Bendersky. RankT5: Fine-tuning T5 for text ranking with ranking losses. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2023. (Cited on page 27.)

[208] S. Zhuang and G. Zuccon. Counterfactual online learning to rank. In *Advances in Information Retrieval: 42nd European Conference on IR Research*, 2020. (Cited on pages 3 and 27.)

[209] L. Zou, H. Mao, X. Chu, J. Tang, W. Ye, S. Wang, and D. Yin. A large scale search dataset for unbiased learning to rank. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, 2022. (Cited on pages 1, 4, 6, 27, 28, 29, 32, 33, 34, 35, 37, 39, 41, 43, 45, 48, 59, 73, 79, 93, 96, 98, 104, 105, and 106.)

[210] L. Zou, H. Mao, X. Chu, W. Ye, C. Hao, S. Wang, D. Yin, J. Tang, A. Sun, and C. Zhang. Pre-training for web search, 2023. URL `https://aistudio.baidu.com/competition/detail/536/0/introduction`. (Cited on pages 29 and 31.)

[211] L. Zou, H. Mao, X. Chu, W. Ye, C. Hao, S. Wang, D. Yin, J. Tang, A. Sun, and C. Zhang. Unbiased learning for web search, 2023. URL `https://aistudio.baidu.com/competition/detail/534/0/introduction`. (Cited on pages 29, 31, and 93.)

Summary

Search engines and recommender systems often rely on implicit user feedback, particularly clicks, to optimize their ranking algorithms. Clicks, however, are a biased signal of user preference. Users can click only the items shown to them and tend to inspect top-ranked items more thoroughly, leading to fewer clicks on lower-ranked but relevant items. Mitigating these statistical biases in click data while optimizing ranking algorithms is the goal of unbiased learning-to-rank (ULTR). Click models are a prevalent ULTR method for bias mitigation by explicitly modeling user behavior, such as which items a user examined, clicked, or was satisfied with. In this thesis, we advance our understanding of neural click models, a modern iteration of traditional click models that integrates neural networks for better generalization, incorporates more complex user feedback, and scales to massive datasets through gradient-based optimization. Over four chapters, we investigate the theoretical properties, real-world effectiveness, and scalability of neural click models for bias correction in ranking.

First, we compare neural click models with inverse propensity scoring, another prevalent bias-correction approach. While both methods are theoretically equivalent under idealized conditions, they exhibit empirical differences in ranking performance when generalizing over feature representations, with neural click models showing a bias toward more examined items.

Second, we evaluate six prominent unbiased learning-to-rank methods on the largest publicly available real-world search engine dataset. While bias-correction methods clearly improve click prediction, they do not consistently improve ranking performance as judged by search engine evaluators, highlighting the need for a deeper understanding of the success and failure conditions of ULTR methods in real-world settings.

Third, we investigate two-tower models, a prevalent neural click model in industry, addressing recent concerns that production systems might negatively affect their ranking performance. We prove that two-tower models require randomized document swaps or overlapping feature distributions to reliably disentangle bias and relevance signals in clicks, and demonstrate that production systems affect only incorrectly specified models. We attribute recent concerns about production systems impacting two-tower models to flawed simulation setups in related work.

Finally, we introduce the open-source framework CLAX, which replaces expectation maximization with gradient-based optimization for training neural click models. CLAX implements ten established click models, achieving comparable click-prediction performance to prevalent EM-based implementations while training orders of magnitude faster, scaling to massive datasets, and enabling modular parameterization.

Through these contributions, this thesis advances both the theoretical understanding and practical implementation of neural click models for bias correction by introducing novel analyses, open-source tools, and datasets.

Samenvatting

Zoekmachines en recommender systems vertrouwen vaak op impliciete feedback van gebruikers, met name kliks, om hun ranking-algoritmes te optimaliseren. Kliks zijn echter een vertekend signaal van gebruikersvoorkeuren. Gebruikers kunnen alleen op de items klikken die aan hen getoond worden en hebben de neiging om hoger gerankte items grondiger te inspecteren, wat leidt tot minder kliks op lager gerankte maar relevante items. Het mitigeren van deze statistische biases in klikdata terwijl ranking-algoritmes geoptimaliseerd worden, is het doel van *unbiased learning-to-rank* (ULTR). Klikmodellen zijn een veelgebruikte ULTR-methode voor bias-mitigatie door expliciet gebruikersgedrag te modelleren, zoals welke items een gebruiker heeft bekeken, aangeklikt of waar de gebruiker tevreden mee was. In dit proefschrift verdiepen we ons begrip van neurale click models, een moderne iteratie van traditionele klikmodellen die neurale netwerken integreert voor betere generalisatie, complexere gebruikersfeedback verwerkt, en schaalt naar massale datasets door middel van gradient-gebaseerde optimalisatie. In vier hoofdstukken onderzoeken we de theoretische eigenschappen, praktische effectiviteit en schaalbaarheid van neurale klikmodellen voor bias-correctie in ranking.

Ten eerste vergelijken we neurale klikmodellen met *inverse propensity scoring*, een andere veelgebruikte bias-correctiebenadering. Hoewel beide methoden theoretisch equivalent zijn onder geïdealiseerde condities, vertonen ze empirische verschillen in ranking-prestaties bij generalisatie over feature-representaties, waarbij neurale klikmodellen een bias tonen richting meer onderzochte items.

Ten tweede evalueren we vijf prominente *unbiased learning-to-rank* methoden op de grootste, publiek beschikbare *real-world* zoekmachine-dataset. Hoewel bias-correctiemethoden duidelijk klik-predictie verbeteren, verbeteren ze niet consistent de ranking-prestaties zoals beoordeeld door zoekmachine-evaluatoren, wat de noodzaak benadrukt van een dieper begrip van de succes- en faalcondities van ULTR-methoden in *real-world* omgevingen.

Ten derde onderzoeken we *two-tower* modellen, een veelgebruikt neuraal klikmodel in de industrie, waarbij we recente zorgen adresseren dat productiesystemen hun ranking-prestaties negatief zouden kunnen beïnvloeden. We bewijzen dat *two-tower* modellen gerandomiseerde omwisseling van documenten of overlappende distributies van kenmerken vereisen om betrouwbaar bias- en relevantiesignalen in kliks te ontwarren, en laten zien dat productiesystemen alleen incorrect gespecificeerde modellen beïnvloeden. We schrijven recente zorgen over productiesystemen die two-tower models beïnvloeden toe aan gebrekkige simulatie-omgevingen in gerelateerd werk.

Ten slotte introduceren we het open-source framework CLAX, dat *expectation maximization* vervangt met gradient-gebaseerde optimalisatie voor het trainen van neurale klikmodellen. CLAX implementeert tien gevestigde klikmodellen en bereikt vergelijkbare klik-predictie-prestaties met veelgebruikte EM-gebaseerde implementaties, terwijl het ordes van grootte sneller traint, schaalt naar massale datasets en modulaire parameterisatie mogelijk maakt.

Door deze bijdragen bevordert dit proefschrift zowel het theoretisch begrip als de praktische implementatie van neurale klikmodellen voor bias-correctie door nieuwe analyses, open-source tools, en datasets te introduceren.