CLAX: Fast and Flexible Neural Click Models in JAX

Philipp Hager*
Booking.com
Amsterdam, The Netherlands
philipp.hager@booking.com

Onno Zoeter
Booking.com
Amsterdam, The Netherlands
onno.zoeter@booking.com

Maarten de Rijke
University of Amsterdam
Amsterdam, The Netherlands
m.derijke@uva.nl

Abstract

CLAX is a JAX-based library that implements classic click models using modern gradient-based optimization. While neural click models have emerged over the past decade, complex click models based on probabilistic graphical models (PGMs) have not systematically adopted gradient-based optimization, preventing practitioners from using modern deep learning frameworks while preserving the interpretability of classic models. CLAX addresses this gap by replacing EM-based optimization with direct gradient-based optimization in a numerically stable manner. The framework's modular design enables the integration of any component, from embeddings and deep networks to custom modules, into classic click models for end-to-end optimization. We demonstrate CLAX's efficiency by running experiments on the full Baidu-ULTR dataset [53] comprising over a billion user sessions in ~ 2 hours on a single GPU, orders of magnitude faster than traditional EM approaches. CLAX implements ten classic click models, serving both industry practitioners seeking to understand user behavior and improve ranking performance at scale and researchers developing new click models. CLAX is available at: https://github.com/philipphager/clax

CCS Concepts

• **Information systems** → **Learning to rank**.

Keywords

Click models, Unbiased learning to rank, JAX

ACM Reference Format:
Philipp Hager, Onno Zoeter, and Maarten de Rijke. 2026. CLAX: Fast and Flexible Neural Click Models in JAX. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '26), July 20–24, 2026, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3805712.3808584

1 Introduction

Click models are integral to web search as tools for understanding and predicting user interactions [9]. Click models are used to determine user behavior on search engine result pages [5, 11, 17, 37], predict clicks in advertising [8, 34, 51], estimate click biases for counterfactual learning-to-rank [44, 46], simulate users [14, 25], evaluate new ranking systems [10], and as ranking models [49].

Listing 1: A minimal example of training a UBM in CLAX.

```python
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)
```

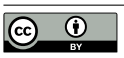
Early click models can be represented as probabilistic graphical models [PGMs, 9] that explicitly model latent variables such as document attractiveness, user satisfaction, and rank examination. Popular models, including the position-based model [PBM, 37], user browsing model [UBM, 17], and dynamic Bayesian network [DBN, 5], are traditionally optimized using expectation maximization (EM). EM optimizes probabilistic models with latent variables by alternately imputing missing values and updating parameters, and is widely used in click modeling libraries such as PyClick[1] or ParClick [27]. The iterative nature of EM scales poorly with dataset size, motivating specialized algorithms [27, 43] and online optimization procedures [33].

Over the last decade, neural click models have emerged along two directions. The first uses neural architectures such as recurrent neural networks [2, 6], attention [52], and graph neural networks [31] to improve click prediction, potentially at the expense of the interpretability of traditional PGMs. The second parameterizes simple PGMs with deep neural networks [13, 49]. A prominent example is the two-tower model, a neural parameterization of the PBM [37], popular for position bias correction in industry [21, 23, 49, 50].

Gradient-based optimization has not been systematically applied to complex PGM-based click models [9], preventing practitioners from using modern deep learning frameworks while preserving the interpretability and theoretical foundations of classic models. Given that simple two-tower models already demonstrate strong empirical performance [21–23, 49], parameterizing more sophisticated models like the DBN [5] and UBM [17] may yield significant benefits.

Building on Google's JAX [3] and Flax NNX [24], 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.

* Work conducted at the University of Amsterdam.



SIGIR '26, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-2599-9/2026/07
https://doi.org/10.1145/3805712.3808584

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

CLAX is designed around modularity and scale. By default, CLAX uses embeddings to represent model components such as query-document attractiveness, making it equivalent to classic click modeling libraries [9]. Listing 1 shows a minimal example of training a UBM with CLAX. CLAX's modular design enables broad customization: linear models, deep networks, deep cross networks [45], or custom Flax modules can be plugged in for end-to-end optimization [24]. This modularity even allows meta-models, e.g., mixture models over multiple click models for datasets where users exhibit different behaviors across sessions [49].

To achieve scale, CLAX uses baseline correction and embedding compression from the deep recommender systems literature [41, 47], combined with JAX's computational efficiency to train on billions of query-document pairs on a single GPU. We demonstrate this by training and evaluating on the complete Baidu-ULTR dataset [53] containing over 1 billion user sessions in under 2 hours.

Our contributions are fourfold:

- We demonstrate that 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 neural components 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 find that neural parameterizations of sophisticated PGMs can surpass the ranking performance of prevalent two-tower models, potentially providing practitioners with a powerful new tool.

CLAX is a tool for practitioners and researchers alike. Practitioners may go beyond two-tower models to cascade-like models. Researchers benefit from the direct optimization of the marginal log-likelihood of many PGMs using SGD and autograd frameworks, simplifying the creation of new models. CLAX's core paradigm is not tied to JAX; all models can be implemented in PyTorch or TensorFlow. We chose JAX for its speed and growing ecosystem [12].

2 Related Work

2.1 Click models

Click models emerged as PGMs for predicting user clicks on search results [9]. Most are extensions of two models: the position-based model [PBM, 37], which assumes users click on documents only after observing their position and finding the document attractive. The cascade model [11] assumes users sequentially examine documents and click on the first relevant item. Extensions to the cascade model include the DCM [20], which assumes a rank-dependent probability of users continuing after their first click, the CCM [19], where continuation depends on a document's attractiveness, and the DBN [5], separating document attraction (before click) and satisfaction (after clicking). The UBM [17] extends the PBM by assuming that examination depends on the last clicked position. CLAX implements seven foundational PGM-based click models and three CTR baselines using gradient-based optimization [9].[2]

2.2 Neural click models

Click models have incorporated neural networks in two ways. First, new click models based on neural architectures, such as recurrent neural networks, graph networks, and attention have emerged [2, 6, 31, 52]. These methods demonstrate strong empirical performance though are less interpretable than PGM-based click models.

Second, neural implementations of classic click models have emerged. Two-tower models based on the PBM are prevalent in industry ranking settings [21, 23, 49], using query-document features and bias features beyond position to predict examination and attractiveness. CLAX provides the logical next step by enabling easy parameterization of more advanced click models.

We are not the first to apply gradient-based optimization to PGM-based click models beyond the PBM. Deffayet et al. [13] implement the PBM, UBM, and DBN using gradient-based optimization in PyTorch while investigating robustness to distributional shift. Their work focuses on robustness analysis rather than validation against EM baselines or providing a general-purpose library. CLAX extends this 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) scaling to massive datasets.

2.3 Existing frameworks

The standard library for click modeling is PyClick [9], using maximum likelihood estimation and EM. PyClick can be accelerated with the PyPy [1] interpreter, but it remains too slow even for medium-scale datasets (see Section 6.1). Specialized libraries have emerged to address this. ParClick [27] is a C++ library that parallelizes EM across multiple CPU cores on a single machine. MassiveClicks [43] extends ParClick to support multi-node and GPU-based training. These libraries enable fast training but are difficult to extend with custom neural modules.

Rather than competing with specialized libraries on speed, CLAX replaces EM with gradient-based optimization using a general-purpose deep learning framework. This enables end-to-end optimization of customizable models while benefiting from JAX's speed through just-in-time compilation and GPU support [3, 24]. CLAX integrates with the JAX ecosystem, including Flax [24] for neural networks, Optax for optimization, and Rax [26] for ranking metrics. Our implementation demonstrates a paradigm that translates to other deep learning frameworks, such as PyTorch or TensorFlow.

3 From EM to Gradient-based Optimization

In the following, we theoretically compare expectation maximization [15] and gradient ascent[3] for inferring parameters in click models, using the position-based model (PBM) [11, 37] as an example. We focus on a single query with documents $d \in D$ at positions $k \in \{1, \dots, K\}$. The PBM assumes a user clicks on document d if they examine position k and find it attractive. 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:

$$P(C = 1 \mid d, k) = P(E = 1 \mid k) \cdot P(A = 1 \mid d) = \theta_k \gamma_d, \qquad (1)$$

[2] We provide the complete derivation of each model and its marginal log-likelihood in our documentation: https://philipphager.github.io/clax/2-models/

[3] Here, we frame the problem as likelihood maximization, although in practice we minimize the negative log-likelihood using gradient descent.

where θ_k is the examination probability of 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 examination or attractiveness. This means we cannot directly optimize the complete-data log-likelihood. Instead, we must maximize the marginal log-likelihood of the data we can actually observe:

$$\mathcal{L}(\theta, \gamma; \mathcal{D}) = \sum_{(d,k,c)\in\mathcal{D}} \left[c \log(\theta_k \gamma_d) + (1-c)\log(1-\theta_k\gamma_d)\right]. \quad (2)$$

3.1 Expectation-maximization (EM) algorithm

EM [15] is a prominent method for optimizing likelihoods with latent variables. Starting from an initial guess of our parameters, EM cycles between two steps until convergence. In the expectation step, we compute posterior expectations of latent variables using current parameters and observed data. In the maximization step, we find new parameters that maximize the expected complete-data log-likelihood. In the following, we showcase EM for the PBM.

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

$$\begin{aligned}
\hat{e}_{d,k,c} &= \mathbb{E}[E \mid c, d, k; \theta^{(t)}, \gamma^{(t)}] \\
&= c \cdot P(E=1 \mid C=1) + (1-c)\cdot P(E=1 \mid C=0) \\
&= c \cdot 1 + (1-c)\cdot \frac{P(C=0 \mid E=1, d, k) P(E=1 \mid k)}{P(C=0 \mid d, k)} \\
&= c + (1-c)\cdot \frac{(1-\gamma_d^{(t)})\theta_k^{(t)}}{1-\theta_k^{(t)}\gamma_d^{(t)}}, \qquad (3) \\
\hat{a}_{d,k,c} &= \mathbb{E}[A \mid c, d, k; \theta^{(t)}, \gamma^{(t)}] \\
&= c + (1-c)\cdot \frac{(1-\theta_k^{(t)})\gamma_d^{(t)}}{1-\theta_k^{(t)}\gamma_d^{(t)}}. \qquad (4)
\end{aligned}$$

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

$$\begin{aligned}
&Q\left(\theta^{(t+1)}, \gamma^{(t+1)} \mid \theta^{(t)}, \gamma^{(t)}\right) = \\
&\sum_{(d,k,c)\in\mathcal{D}} \left[\hat{e}_{d,k,c} \log(\theta_k^{(t+1)}) + (1-\hat{e}_{d,k,c})\log(1-\theta_k^{(t+1)})\right] + \\
&\sum_{(d,k,c)\in\mathcal{D}} \left[\hat{a}_{d,k,c} \log(\gamma_d^{(t+1)}) + (1-\hat{a}_{d,k,c})\log(1-\gamma_d^{(t+1)})\right].
\end{aligned} \quad (5)$$

Note that the Q-function is commonly much simpler than our marginal log-likelihood. For the PBM, it decouples the estimation of θ and γ. Taking derivatives with respect to each parameter, setting to zero, and solving yields the following closed-form updates:

$$\theta_k^{(t+1)} = \frac{\sum_{(d,k',c)\in\mathcal{D},k'=k} \hat{e}_{d,k,c}}{\sum_{(d,k',c)\in\mathcal{D},k'=k} 1}, \quad \gamma_d^{(t+1)} = \frac{\sum_{(d',k,c)\in\mathcal{D},d'=d} \hat{a}_{d,k,c}}{\sum_{(d',k,c)\in\mathcal{D},d'=d} 1}. \quad (6)$$

3.2 Gradient-based optimization

An alternative to EM is to optimize the marginal log-likelihood $\mathcal{L}$ in Eq. 2 using gradient-based methods. This involves computing the partial derivative of $\mathcal{L}$ w.r.t. each parameter and taking a step in the direction of the gradient [38]:

$$\frac{\partial \mathcal{L}}{\partial \gamma_d} = \sum_{(d',k,c)\in\mathcal{D},d'=d} \left(\frac{c}{\gamma_d} - \frac{(1-c)\theta_k}{1-\theta_k\gamma_d}\right), \quad (7)$$

$$\frac{\partial \mathcal{L}}{\partial \theta_k} = \sum_{(d,k',c)\in\mathcal{D},k'=k} \left(\frac{c}{\theta_k} - \frac{(1-c)\gamma_d}{1-\theta_k\gamma_d}\right). \quad (8)$$

We update model parameters iteratively using learning rate η:

$$\theta_k^{(t+1)} = \theta_k^{(t)} + \eta \frac{\partial \mathcal{L}}{\partial \theta_k} \quad \text{and} \quad \gamma_d^{(t+1)} = \gamma_d^{(t)} + \eta \frac{\partial \mathcal{L}}{\partial \gamma_d}. \quad (9)$$

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 [29, 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 and circumvents the complexity of directly optimizing the marginal likelihood by optimizing a simpler auxiliary function [15, 39, 48]. However, EM can be slow to converge with high missing information [15, 48] and classical implementations require full dataset passes, motivating online and stochastic variants [4, 35, 42].

Gradient-based methods offer different trade-offs, using general-purpose optimization advances (e.g., adaptive learning rates [16, 28, 32], momentum [36]) that can lead to fast convergence, without monotonic improvement guarantees. Their advantage lies in mini-batch processing, which scales well to large datasets and modern parallel hardware. For click models with tractable marginal log-likelihoods, automatic differentiation eliminates manual E and M step derivations while offering computational efficiency gains.

The relationship between EM and gradient methods runs deeper than their shared objective: the gradient of the expected complete-data log-likelihood (Q-function), evaluated at current parameter estimates, equals the gradient of the marginal log-likelihood [39]:

$$\nabla Q(\theta, \gamma \mid \theta^{(t)}, \gamma^{(t)})\Big|_{\theta=\theta^{(t)},\gamma=\gamma^{(t)}} = \nabla \mathcal{L}(\theta, \gamma)\Big|_{\theta=\theta^{(t)},\gamma=\gamma^{(t)}}. \quad (10)$$

In fact, EM can be viewed as gradient ascent with an adaptive, parameter-dependent step size [39]. We verify this equality explicitly for the PBM. The gradient of the auxiliary function Q with respect to γ_d, evaluated at $\theta_k = \theta_k^{(t)}$ and $\gamma_d = \gamma_d^{(t)}$, is:

$$\frac{\partial Q\left(\theta_k, \gamma_d \mid \theta_k, \gamma_d\right)}{\partial \gamma_d} = \sum_{(d',k,c)\in\mathcal{D},d'=d} \left(\frac{c}{\gamma_d} - \frac{(1-c)\theta_k}{1-\theta_k\gamma_d}\right) = \frac{\partial \mathcal{L}}{\partial \gamma_d}. \quad (11)$$

While we verify Eq. 10 only for the PBM for brevity, the fundamental property holds for any click model optimized using EM, including complex models like the DCM, UBM, and DBN [39].

This has important practical consequences. If one implements EM using the most recent parameters in the E-step and performs only a single gradient update for a batch the M-step, EM effectively becomes direct gradient-based maximization of the marginal log-likelihood. E.g., the TensorFlow Ranking library implements an EM-based objective for the PBM [46],[4] but their gradient-based optimization produces the same gradient for each batch as directly minimizing the binary cross-entropy loss in Eq. 2, demonstrating

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

Listing 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))
```

that some practitioners have implicitly adopted direct gradient-based optimization while framing it as EM. Nevertheless, classical implementations typically differ: EM may perform full maximization in each M-step or aggregate the entire dataset in the E-step, while gradient-based methods exploit stochasticity and adaptive optimization. Gradient-based optimization is a theoretically sound alternative to EM for PGM-based click models with tractable marginal likelihoods, motivating CLAX, a JAX-based library for flexible neural parameterization and scalable optimization.

4 An Overview of CLAX

CLAX builds on three principles: (i) numerically stable gradient-based log-likelihood optimization; (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 enhance numerical stability by performing all calculations with log-probabilities.[5]

4.1 The CLAX model API

All models in CLAX share a unified interface of five methods that accept a batch of data, as demonstrated in Listing 2. A batch in CLAX is a Python dictionary of 2D NumPy arrays of shape (batch size, max. positions). By default, CLAX expects arrays of document positions, query-document-ids, clicks, and a binary mask indicating valid (non-padded) entries. Padding queries to a fixed shape enables vectorizing operations across variable-length sessions and reduces JIT recompilation in JAX [3, 26]. Each CLAX model implements five methods:

- `compute_loss(...)` Computes the training objective, typically the negative log-likelihood of observed clicks, but can be customized.
- `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 clicks at previous ranks: $\hat{c} = P(C = 1 \mid d, k, c_{<k})$.
- `predict_relevance(...)` Returns a ranking scores for each documents, typically the document attractiveness.
- `sample(...)` Samples clicks and latent variables (e.g., examination, attractiveness, satisfaction) on the current bactch.

[5] For interested readers, we introduce standard operations between log-probabilities in our documentation: https://philipphager.github.io/clax/4-numerical-stability/

Listing 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),
)
```

4.2 Flexible parameterization

By decoupling a click model's structure from its parameterization, CLAX enables flexible model composition with embeddings, neural networks, or custom Flax modules.

CLAX uses embedding tables by default, allocating separate parameters for model components (e.g., attractiveness per query-document pair). CLAX optionally uses a shared baseline parameter so embeddings encode offsets from a global embedding rather than absolute values. Parameters start at the baseline and gradually adapt with more observations, improving predictions on long-tailed data.

To reduce the memory footprint of embedding tables, CLAX provides two embedding compression techniques: (i) The hashing-trick [47] maps multiple indices to the same embedding using hash functions and (ii) the quotient-remainder trick [41] 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 3 shows how to configure the hashing-trick for the click chain model (CCM). Section 6.2 evaluates both compression techniques and demonstrates training on datasets with billions of query-document pairs on a single GPU.

In many cases, allocating separate embedding parameters for models might not be optimal. Instead, we may want to generalize over shared feature representations, such as two-tower models that use feature representations for examination and attractiveness [21, 49, 50]. CLAX supports configuring any model parameter to use features, with built-in support for linear layers, deep-neural networks, and DeepCrossV2 networks [45]. Listing 4 shows a two-tower model using a linear combination of bias features and a DeepCrossV2 network to predict attractiveness:

Listing 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),
)
```

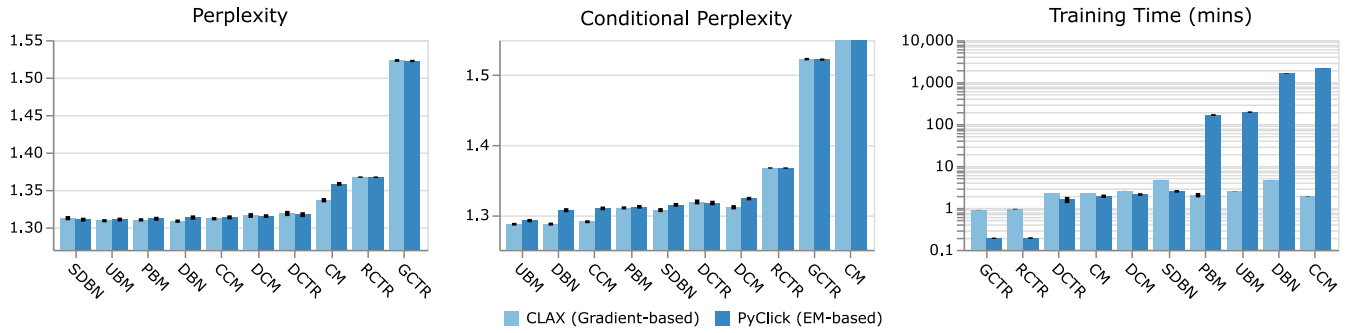


Figure 1: CLAX matches or exceeds the click predictions of PyClick over three folds of 10M training sessions on WSCD-2012.

Finally, 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 the examples in our online repository.

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. Building on [49], we implement an example mixture model that can be used like any other CLAX model. A distribution over a collection of click models is learned, to account for different browsing patterns across sessions. Listing 5 shows how to implement a mixture distribution over a PBM and DBN model.

Yan et al. [49] share attractiveness parameters between 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. We evaluate a mixture model as part of our experiments in Section 6.3.

Listing 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])
```

4.4 Evaluation

Two aspects of click models are commonly evaluated [9, 18]: click prediction and ranking quality. For click prediction, CLAX implements three standard metrics: log-likelihood, conditional, and unconditional perplexity. Conditional perplexity measures how well a model fits observed data by incorporating previous clicks in the session (using conditional click predictions $\hat{c} = P(C = 1 \mid d, k, c_{<k})$), while unconditional perplexity reflects prediction accuracy on completely unseen rankings without session context. This distinction matters when evaluating models like the UBM [17] that adapt predictions based on previous clicks in the same session. Conditional perplexity assumes top-down browsing behavior; if this assumption is violated, unconditional perplexity is preferable. For ranking evaluation, CLAX uses metrics from Rax [26], a JAX library for learning-to-rank, supporting standard metrics such as DCG, MRR, and AP computed against expert relevance judgments. We list metric definitions and usage examples in our documentation: https://philipphager.github.io/clax/3-metrics/

5 Experimental Setup

We conduct experiments in three settings. (i) We compare CLAX models to EM-based counterparts from PyClick. (ii) We scale CLAX to large datasets and investigate the effects of embedding compression. (iii) We evaluate CLAX models as unbiased ranking models.

5.1 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 [9, 40]. 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 [53]. 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 [7, 22, 30, 53, 54]. For ranking performance comparison, we use the subset of Baidu-ULTR from [22] with pre-computed 768-dimensional MonoBERT features for 2,372,947 sessions.[6] We publish all preprocessed datasets and efficient dataloaders using Apache Parquet under: https://huggingface.co/datasets/philipphager/clax-datasets.

5.2 Implementation

All CLAX experiments use the default trainer with the AdamW optimizer [32] (learning rate 0.003, weight decay 0.0001) over 100

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

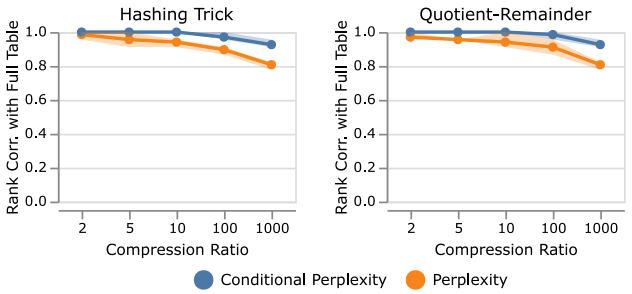


Figure 2: Kendall's τ between ranking models trained with and without embedding compression on WSCD-2012.

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 in our code repository: https://github.com/philipphager/clax.

6 Results

6.1 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. Fig. 1 compares the click prediction performance and training times across both libraries. 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. CLAX sometimes achieves better conditional perplexity despite both libraries optimizing the same objectives. 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 requires from 172 minutes (PBM) to 36 hours (CCM).[7] In contrast, all CLAX models complete training in under 5 minutes. Overall, 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 a specific dataset.

[7] The DCM in PyClick is actually a simplified DCM (SDCM) to use faster MLE while CLAX implements the original latent-variable version [20].

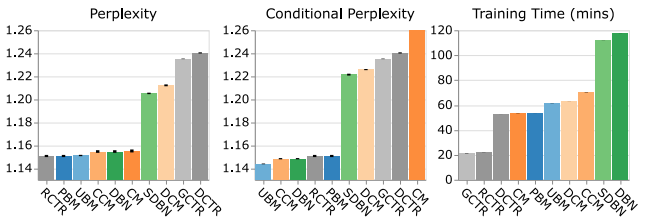


Figure 3: Embedding-based CLAX models on the Baidu-ULTR dataset (three folds of 800M/200M/200M sessions for training, validation, testing) [53]. All models complete training in under 2 hours using the hashing-trick with 10x compression.

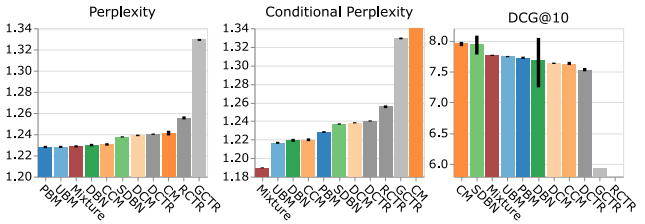


Figure 4: CLAX models generalizing over BERT features on the Baidu-ULTR-UVA dataset [22] using a deep-cross network achieve strong ranking performance and a different model fit compared to embedding-based models.

6.2 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 vs. 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×. Fig. 2 compares click prediction rankings for models trained with and without compression using Kendall's tau. The compression methods behave similarly, maintaining remarkably high correlation up to very high compression ratios of 10–100×. Unconditional perplexity is more susceptible to compression. As Fig. 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. Compression reduces overall click prediction performance (higher perplexity). As some CTR baselines, e.g., the RCTR and GCTR, do not use compression, comparing compressed and uncompressed models can lead to wrong conclusions at high compression rates. Beyond reducing the memory footprint, compression decreases the average training time from 14 minutes for uncompressed models to around four minutes for 10x compression and higher.

Second, we evaluate scale by training CLAX models using the hashing-trick with 10x compression on three folds of the full Baidu-ULTR dataset containing over 1B user sessions. Fig. 3 shows the resulting models, all trained in under 2 hours.

6.3 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. Fig. 4 shows the results on the Baidu-ULTR-UVA subset [22]. 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 [22]. Nevertheless, cascade-based models achieve strong ranking performance, comparable to listwise LTR loss functions trained in previous work [22, Fig. 3]. Thus, complex click models can be effective ranking models in real-world settings.

We also assess the effectiveness of our mixture model, which combines a PBM, DCTR, and GCTR model following the setup from [49] but excludes the RCTR as it cannot be applied to the Baidu-ULTR test set (Fig. 4). The mixture model achieves better click prediction and ranking performance than its individual components.

7 Conclusion

We have introduced CLAX, the first JAX-based click modeling library enabling end-to-end gradient-based optimization for PGM-based click models. CLAX demonstrates that gradient-based optimization can replace EM for training click models while achieving comparable performance. The framework provides orders of magnitude speedup over established implementations, training and evaluating on over 1B user sessions in ~2 hours on a single GPU.

CLAX's modular design decouples model logic from parameterization, supporting embeddings, deep networks, and custom modules. Through embedding compression techniques, the framework scales to billions of query-document pairs. Our experiments show that neural parameterizations of complex PGMs and mixture models can surpass widely-used two-tower models in ranking tasks.

The CLAX framework serves both industry practitioners seeking to understand user behavior at scale and researchers developing new click models. All code and datasets are open-source, enabling reproducible research and practical adoption.

Our implementation currently lacks support for sparse embeddings, which can negatively impact performance on CPUs. In the future, we will support sparse embeddings, neural click models beyond classical PGMs, and distributed training across GPUs.

Acknowledgments

We thank David Rohde, Oscar Ramirez, Mathijs de Jong, Ahmed Khaili, and Shashank Gupta for their invaluable feedback. This research was supported by the Mercury Machine Learning Lab created by TU Delft, the University of Amsterdam, and Booking.com. Maarten de Rijke was supported by the Dutch Research Council (NWO), project nrs 024.004.022, NWA.1389.20.183, and KICH3.LTP.20.006, and the European Union's Horizon Europe program under grant agreement No 101070212. All content represents the opinion of the authors, which their employers or sponsors do not necessarily endorse.

References

[1] Carl Friedrich Bolz, Antonio Cuni, Maciej Fijalkowski, and Armin Rigo. 2009. Tracing the meta-level: PyPy's tracing JIT compiler. In *Proceedings of the 4th Workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems.*

[2] Alexey Borisov, Ilya Markov, Maarten de Rijke, and Pavel Serdyukov. 2016. A Neural Click Model for Web Search. In *Proceedings of the 25th International Conference on World Wide Web.*

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2022. *JAX: Composable transformations of Python+NumPy programs.* http://github.com/google/jax Version 0.3.13.

[4] Olivier Cappé. 2011. Online Expectation Maximisation. In *Mixtures: Estimation and Applications.* Wiley Online Library, 31–53.

[5] Olivier Chapelle and Ya Zhang. 2009. A Dynamic Bayesian Network Click Model for Web Search Ranking. In *Proceedings of the 18th International Conference on World Wide Web.*

[6] Jia Chen, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. 2020. A Context-Aware Click Model for Web Search. In *Proceedings of the 13th International Conference on Web Search and Data Mining.*

[7] Xiaoshu Chen, Xiangsheng Li, Kunliang Wei, Bin Hu, Lei Jiang, Zeqian Huang, and Zhanhui Kang. 2023. Multi-Feature Integration for Perception-Dependent Examination-Bias Estimation. In *Proceedings of the 16th ACM International Conference on Web Search and Data Mining.*

[8] Ye Chen and Tak W. Yan. 2012. Position-normalized Click Prediction in Search Advertising. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

[9] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. 2015. *Click Models for Web Search.* Morgan & Claypool.

[10] Aleksandr Chuklin, Pavel Serdyukov, and Maarten de Rijke. 2013. Click Model-based Information Retrieval Metrics. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval.*

[11] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. 2008. An Experimental Comparison of Click Position-Bias Models. In *Proceedings of the 2008 International Conference on Web Search and Data Mining.*

[12] DeepMind, Igor Babuschkin, Kate Baumli, Alison Bell, Surya Bhupatiraju, Jake Bruce, Peter Buchlovsky, David Budden, Trevor Cai, Aidan Clark, Ivo Danihelka, Antoine Dedieu, Claudio Fantacci, Jonathan Godwin, Chris Jones, Ross Hemsley, Tom Hennigan, Matteo Hessel, Shaobo Hou, Steven Kapturowski, Thomas Keck, Iurii Kemaev, Michael King, Markus Kunesch, Lena Martens, Hamza Merzic, Vladimir Mikulik, Tamara Norman, George Papamakarios, John Quan, Roman Ring, Francisco Ruiz, Alvaro Sanchez, Laurent Sartran, Rosalia Schneider, Eren Sezener, Stephen Spencer, Srivatsan Srinivasan, Miloš Stanojević, Wojciech Stokowiec, Luyu Wang, Guangyao Zhou, and Fabio Viola. 2020. *The DeepMind JAX Ecosystem.*

[13] Romain Deffayet, Jean-Michel Renders, and Maarten de Rijke. 2023. Evaluating the Robustness of Click Models to Policy Distributional Shift. *ACM Transactions on Information Systems (TOIS)* 41, 4, Article 84 (2023).

[14] Romain Deffayet, Thibaut Thonet, Dongyoon Hwang, Vassilissa Lehoux, Jean-Michel Renders, and Maarten de Rijke. 2024. SARDINE: Simulator for Automated Recommendation in Dynamic and Interactive Environments. *ACM Transactions on Recommender Systems (TORS)* 2, 3, Article 25 (2024).

[15] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. 1977. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society* 39, 1 (1977), 1–38.

[16] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research* 12 (2011), 2121–2159.

[17] Georges E. Dupret and Benjamin Piwowarski. 2008. 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.*

[18] Artem Grotov, Aleksandr Chuklin, Ilya Markov, Luka Stout, Finde Xumara, and Maarten de Rijke. 2015. A Comparative Study of Click Models for Web Search. In *Proceedings of the 6th International Conference on Experimental IR Meets Multilinguality, Multimodality, and Interaction.*

[19] Fan Guo, Chao Liu, Anitha Kannan, Tom Minka, Michael Taylor, Yi-Min Wang, and Christos Faloutsos. 2009. Click Chain Model in Web Search. In *Proceedings of the 18th International Conference on World Wide Web.*

[20] Fan Guo, Chao Liu, and Yi Min Wang. 2009. Efficient Multiple-click Models in Web Search. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining.*

[21] Huifeng Guo, Jinkai Yu, Qing Liu, Ruiming Tang, and Yuzhou Zhang. 2019. PAL: A Position-Bias Aware Learning Framework for CTR Prediction in Live Recommender Systems. In *Proceedings of the 13th ACM Conference on Recommender Systems.*

[22] Philipp Hager, Romain Deffayet, Jean-Michel Renders, Onno Zoeter, and Maarten de Rijke. 2024. 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*.

[23] Malay Haldar, Prashant Ramanathan, Tyler Sax, Mustafa Abdool, Lanbo Zhang, Aamir Mansawala, Shulin Yang, Bradley Turnbull, and Junshuo Liao. 2020. Improving Deep Learning for Airbnb Search. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[24] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. 2024. *Flax: A Neural Network Library and Ecosystem for JAX*. http://github.com/google/flax Version 0.10.5.

[25] Jin Huang, Harrie Oosterhuis, Maarten de Rijke, and Herke van Hoof. 2020. 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*.

[26] Rolf Jagerman, Xuanhui Wang, Honglei Zhuang, Zhen Qin, Michael Bendersky, and Marc Najork. 2022. Rax: Composable Learning-to-Rank Using JAX. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

[27] Pooya Khandel, Ilya Markov, Andrew Yates, and Ana-Lucia Varbanescu. 2022. ParClick: A Scalable Algorithm for EM-based Click Models. In *Proceedings of the ACM Web Conference*.

[28] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980* (2014).

[29] Daphne Koller and Nir Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. MIT press.

[30] Xiangsheng Li, Xiaoshu Chen, Kunliang Wei, Bin Hu, Lei Jiang, Zeqian Huang, and Zhanhui Kang. 2023. 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*.

[31] Jianghao Lin, Weiwen Liu, Xinyi Dai, Weinan Zhang, Shuai Li, Ruiming Tang, Xiuqiang He, Jianye Hao, and Yong Yu. 2021. A Graph-Enhanced Click Model for Web Search. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

[32] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101* (2019).

[33] Ilya Markov, Alexey Borisov, and Maarten de Rijke. 2017. Online Expectation-Maximization for Click Models. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*.

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

[35] Radford M. Neal and Geoffrey E. Hinton. 1998. A View of the EM Algorithm that Justifies Incremental, Sparse, and Other Variants. In *Learning in Graphical Models*. Springer, 355–368.

[36] Ning Qian. 1999. On the Momentum Term in Gradient Descent Learning Algorithms. *Neural Networks* 12, 1 (1999), 145–151.

[37] Matthew Richardson, Ewa Dominowska, and Robert Ragno. 2007. Predicting Clicks: Estimating the Click-through Rate for New Ads. In *Proceedings of the 16th International Conference on World Wide Web*.

[38] Sebastian Ruder. 2017. An Overview of Gradient Descent Optimization algorithms. *arXiv preprint arXiv:1609.04747* (2017).

[39] Ruslan Salakhutdinov, Sam T. Roweis, and Zoubin Ghahramani. 2003. Optimization with EM and Expectation-Conjugate-Gradient. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*.

[40] Pavel Serdyukov, Nick Craswell, and Georges Dupret. 2012. WSCD 2012: Workshop on Web Search Click Data 2012. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining*.

[41] Hao-Jun Michael Shi, Dheevatsa Mudigere, Maxim Naumov, and Jiyan Yang. 2020. Compositional Embeddings using Complementary Partitions for Memory-efficient Recommendation Systems. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*.

[42] Bo Thiesson, Christopher Meek, and David Heckerman. 2001. Accelerating EM for large databases. *Machine Learning* 45, 3 (2001), 279–299.

[43] Skip Thijssen, Pooya Khandel, Andrew Yates, and Ana-Lucia Varbanescu. 2023. MassiveClicks: A Massively-Parallel Framework for Efficient Click Models Training. In *European Conference on Parallel Processing*.

[44] Ali Vardasbi, Harrie Oosterhuis, and Maarten de Rijke. 2020. 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*.

[45] Ruoxi Wang, Rakesh Shivanna, Derek Cheng, Sagar Jain, Dong Lin, Lichan Hong, and Ed Chi. 2021. DCN V2: Improved Deep & Cross Network and Practical Lessons for Web-scale Learning to Rank Systems. In *Proceedings of the Web Conference*.

[46] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. 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*.

[47] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature Hashing for Large Scale Multitask Learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*.

[48] C. F. Jeff Wu. 1983. On the Convergence Properties of the EM Algorithm. *The Annals of Statistics* 11, 1 (1983), 95–103.

[49] Le Yan, Zhen Qin, Honglei Zhuang, Xuanhui Wang, Michael Bendersky, and Marc Najork. 2022. 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*.

[50] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending What Video to Watch Next: A Multitask Ranking System. In *Proceedings of the 13th ACM Conference on Recommender Systems*.

[51] Zeyuan Allen Zhu, Weizhu Chen, Tom Minka, Chenguang Zhu, and Zheng Chen. 2010. A Novel Click Model and its Applications to Online Advertising. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*.

[52] Honglei Zhuang, Zhen Qin, Xuanhui Wang, Michael Bendersky, Xinyu Qian, Po Hu, and Dan Chary Chen. 2021. Cross-Positional Attention for Debiasing Clicks. In *Proceedings of the Web Conference*.

[53] Lixin Zou, Haitao Mao, Xiaokai Chu, Jiliang Tang, Wenwen Ye, Shuaiqiang Wang, and Dawei Yin. 2022. A Large Scale Search Dataset for Unbiased Learning to Rank. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*.

[54] Lixin Zou, Haitao Mao, Xiaokai Chu, Wenwen Ye, Changying Hao, Shuaiqiang Wang, Dawei Yin, Jiliang Tang, Aixin Sun, and Ce Zhang. 2023. Unbiased Learning for Web Search. https://aistudio.baidu.com/competition/detail/534/0/introduction