CLAX: Fast and Flexible Neural Click Models in JAX

Philipp Hager
University of Amsterdam
Amsterdam, The Netherlands
p.k.hager@uva.nl

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 leveraging 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 [60] 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

1 Introduction

Click models are an integral part of web search and recommender systems, serving as fundamental tools for understanding and predicting user interactions [10]. Click models are used to determine user behavior on search engine result pages [6, 12, 18, 44], predict clicks in advertising [9, 40, 58], estimate click biases for counterfactual learning-to-rank methods [51, 53], simulate users in reinforcement learning [15, 28], evaluate new ranking systems [11], and are directly used as ranking models [56].

Early click models can be represented as probabilistic graphical models (PGMs) [10] that explicitly model variables such as document attractiveness [12, 44], user satisfaction [6, 21], and rank examination [18, 44] that might influence user behavior. Since most of these variables are latent, popular models including the position-based model (PBM) [44], user browsing model (UBM) [18], and

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)
```

dynamic Bayesian network (DBN) of [6] 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 [31]. The iterative nature of EM, however, scales poorly with dataset size, motivating specialized algorithms [31, 50] and online optimization procedures [39].

Over the last decade, neural click models have emerged along two directions. The first uses neural architectures such as recurrent neural networks [2, 7], attention [59], or graph neural networks [36] to improve click prediction and ranking performance, potentially sacrificing the interpretability that made traditional PGMs valuable for understanding users. The second branch parameterizes simple PGMs with deep neural networks [14, 56]. A prominent example is the two-tower model, a neural parameterization of the PBM [44], popular for position bias correction in industry [23, 25, 56, 57].

However, a significant gap remains: *the paradigm shift to gradient-based optimization has not been systematically applied to more complex PGM-based click models [10], preventing practitioners from leveraging 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 [23, 25, 56], parameterizing more sophisticated models like the DBN [6] and UBM [18] may yield significant improvements.

Modern frameworks such as JAX [3] are compelling for addressing this gap. JAX's functional programming and automatic differentiation are well-suited for complex probabilistic models, while just-in-time (JIT) compilation and vectorization enable efficient computation on large datasets. And high-level deep learning libraries, such as Flax NNX [26], make JAX more accessible.

We introduce CLAX: A neural click modeling library bridging traditional PGMs with modern gradient-based optimization in JAX. CLAX replaces EM with direct optimization of marginal log-likelihoods using gradient descent in a numerically stable way.

---

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

CLAX is modular and allows broad customization. By default, CLAX uses embeddings, e.g., to represent the attractiveness of individual query-document pairs, making it equivalent to classic click modeling libraries. Listing 1 shows an example of training a UBM with CLAX. Building on embeddings, CLAX uses methods from the deep recommender systems literature to compress large embedding tables [48, 54], enabling scaling to billions of query-document pairs on a single GPU. Importantly, CLAX enables flexible customizations beyond embeddings, including linear models, deep networks, deep-cross networks, and custom FLAX models. 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. [56] and training a mixture model over multiple click models for datasets where users exhibit different behaviors across sessions. To summarize, our contributions are fourfold:

- We demonstrate that direct gradient-based optimization can replace EM for training traditional PGM-based click models and achieves comparable empirical performance.
- We introduce CLAX, the first 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 1B 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, providing a powerful new tool for practitioners.

CLAX is meant as a library 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 computational speed and growing ecosystem.

Below, we first cover related work on click modeling, implementations in the field, and the JAX ecosystem. Section 3 compares gradient-based with EM-based optimization for click models. Section 4 provides an overview of the CLAX library and Section 5 discusses the basics of its numerically stable implementation. Lastly, we evaluate CLAX empirically in Sections 6 and 7.

2 Related Work

2.1 Click models

Click models emerged as probabilistic graphical models (PGMs) to predict user clicks on search engine result pages [10]. Most click models are extensions of two models. The position-based model (PBM) [44] 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 [12] 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) [22] 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) [21] extends this idea by assuming users continue after a click based on the attractiveness of the current document. The dynamic Bayesian network (DBN) [6] 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) [18] 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. [10], using gradient-based optimization. We provide the complete derivation of each model and its marginal log-likelihood in Appendix A.

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 [2] or the CACM [7] predict clicks using recurrent neural networks. The XPA model by Zhuang et al. [59] uses attention to capture interactions between documents in a grid layout. And the GraphCM [36] 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 [23, 25, 56], 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 PGM-based click models.

2.3 Frameworks for click modeling

The standard library for click modeling is PyClick,[2] published by Chuklin et al. [10], 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 7). To address these performance limitations, specialized libraries have emerged. ParClick [31][4] is a C++ library that parallelizes EM across multiple CPU cores on a single machine. MassiveClicks [50][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 [3, 26]. Our JAX implementation demonstrates a paradigm that can be translated to other deep learning frameworks.

Lastly, we acknowledge that we are not the first to apply gradient-based optimization to PGM-based click models beyond PBM. Deffayet et al. [14] implement the PBM, UBM, and DBN using gradient-based optimization in PyTorch while investigating click model robustness to distributional shift. However, since their work focused on robustness analysis rather than providing a general-purpose library, CLAX offers a more extensible API and improved numerical stability through computation in log-probability space.

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 [3]. Unlike monolithic frameworks such as PyTorch or TensorFlow, JAX adopts a modular ecosystem approach where functionality is distributed across specialized libraries [13]. The base JAX library provides a NumPy-compatible interface and fundamental transformations, while additional libraries offer domain-specific functionality: Flax NNX [26] and Haiku [27] are neural network libraries, Optax supplies optimizers, Chex offers testing utilities, and Distrax provides probability distributions. This modular design enables research communities to develop specialized tools for their respective field [4, 32, 43]. Within information retrieval, Rax [29] 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.

3 Comparing Expectation Maximization and Gradient-based Optimization

Next, we compare the expectation maximization algorithm [16] and gradient ascent[6] for inferring parameters in click models, using the example of the position-based model (PBM) [12, 44]. 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:

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

---

[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 work differentiate between users being attracted to a document snippet and being satisfied with a document after clicking, we follow Chuklin et al. [10] and use the more precise terminology of "attractiveness."

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:

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

3.1 Expectation-maximization (EM) algorithm

A prominent method for optimizing likelihoods with latent variables is the EM algorithm [16]. Starting from an initial guess for our model parameters, EM cycles between 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:

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

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

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 (Q-function):

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

Note that the 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 Q

with respect to each parameter, setting it to zero, and solving, we obtain closed-form update rules for the PBM:

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

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

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.

3.2 Gradient-based optimization

An alternative to EM is to optimize the marginal log-likelihood $\mathcal{L}$ in Eq. 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 [45]:

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

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

We update model parameters iteratively using learning rate η:

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

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 [34, 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 [16, 46, 55]. However, EM can be slow to converge in settings with high missing information [16, 55], and classical implementations require full dataset passes, a limitation that motivated online and stochastic variants [5, 41, 49].

Gradient-based methods offer different trade-offs, using general-purpose optimization advances (e.g., adaptive learning rates [17, 33, 37], momentum [42]) 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 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 (Q-function), evaluated at the current parameter estimates, is equal to the gradient of the marginal log-likelihood [46]:

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

For the PBM, we can verify this equality explicitly. The gradient of the auxiliary function with respect to γ_d, evaluated at $\theta_k = \theta_k^{(t)}$ and $\gamma_d = \gamma_d^{(t)}$, simplifies to:

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

This theoretical connection has practical implications. When EM is implemented using gradient-based optimization in the M-step, and the E-step uses the most recent model parameters, taking a single gradient step during maximization makes the method equivalent to direct gradient-based optimization of the marginal log-likelihood.[8] More broadly, EM can be interpreted as gradient ascent with adaptive, parameter-dependent step size [46]. However, we emphasize that classical implementations of both approaches differ: EM may perform full maximization in each M-step or aggregate the entire dataset in the E-step, while gradient-based methods can exploit stochasticity and adaptive optimization.

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.

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 2 below:

Listing 2: The CLAX model API.

```
1 batch = {
2     "positions": [[1, 2, 3, ...]],
3     "query_doc_ids": [[101, 205, 847, ...]],
4     "clicks": [[0., 1., 0., ...]],
5     "mask": [[True, True, True, ...]],
6 }
7
8 loss = model.compute_loss(batch)
9 log_probs = model.predict_clicks(batch)
10 cond_log_probs = model.predict_conditional_clicks(batch)
11 relevance_scores = model.predict_relevance(batch)
12 output = model.sample(batch, rngs=nnx.Rngs(42))
```

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 starting at 1, query-document-ids, clicks, and a binary mask. Both the variables and their names depend on the specific 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

---

[8]For example, the EM-based implementation of the RegressionEM [53] click model in TensorFlow ranking leads to the same gradient as the binary cross-entropy loss in Eq. 2, https://www.tensorflow.org/ranking/api_docs/python/tfr/keras/losses/ClickEMLoss.

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 [3, 29]. Each CLAX model implements five methods:

- `compute_loss(...)` Computes the training objective, typically the negative log-likelihood of observed clicks, but custom models may implement alternative 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.

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. 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 [18] in Listing 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 in a table, 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.

Listing 3: Adding hashing and baseline correction to a CCM.

```
1  model = ClickChainModel(
2      attraction=EmbeddingParameterConfig(
3          use_feature="query_doc_ids",
4          parameters=100_000_000,
5          add_baseline=True,
6          embedding_fn=partial(
7              HashEmbedding,
8              compression_ratio=10
9          ),
10      ),
11      rngs=nnx.Rngs(42),
12  )
```

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 provides two embedding compressions: (i) The hashing-trick [54] maps multiple indices to the same embedding using hash functions, reducing table size at the cost of hash collisions. (ii) The quotient-remainder trick [48] 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). A compression ratio of ten means that the method will hash 100M embeddings down to 10M embedding parameters. Section 7 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., because the dataset is too sparse. Instead, we might want to generalize over shared feature representations, like two-tower models that use feature representations for examination and attractiveness parameters [23, 56, 57]. 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 [52].

Listing 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 4: Building a two-tower model in CLAX.

```
1  model = PositionBasedModel(
2      examination=LinearParameterConfig(
3          use_feature="bias_features",
4          features=8,
5      ),
6      attraction=DeepCrossParameterConfig(
7          use_feature="query_doc_features",
8          features=136,
9          cross_layers=2,
10          deep_layers=2,
11          combination=Combination.STACKED,
12      ),
13      rngs=nnx.Rngs(42),
14  )
```

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.

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

---

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

Yan et al. [56], which uses the EM algorithm to learn a distribution over multiple click models, capturing different user behaviors across sessions. This approach 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, limiting click prediction on unseen rankings [56].

CLAX offers a mixture model that extends the idea of Yan et al. [56] 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:

$$\mathcal{LL}_{\text{mixture}}(s) = -\log \left(\sum_{m \in M} P(m) \exp(-\mathcal{LL}_m(s)/\tau) \right), \quad (12)$$

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 shows how to learn a mixture distribution over a PBM and DBN model:

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])
```

Note that Yan et al. [56] 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. We evaluate a mixture model as part of our experiments in Section 7.

4.4 Evaluation

More critical than training models is evaluating click models. Typically, we assess two main aspects of click models [10, 20]: 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:

$$\text{LL}(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(d,k,c) \in \mathcal{D}} \left[c \log \hat{c} + (1-c) \log (1 - \hat{c}) \right], \quad (13)$$

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. 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, as the model is as uncertain as a coin flip. Perplexity is defined as:

$$\text{PPL}(\mathcal{D}) = 2^{-\frac{1}{|\mathcal{D}|} \sum_{(d,k,c) \in \mathcal{D}} \left[c \log_2 \hat{c} + (1-c) \log_2 (1-\hat{c}) \right]}. \quad (14)$$

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 [18]). 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 6 below:

Listing 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()
```

Ranking metrics. A second aspect commonly evaluated of click models is their ranking performance, 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 [29]:

---

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

Listing 7: Support for Rax-based ranking metrics.

```
1  metrics = MultiMetric({
2      "dcg@10": RaxMetric(rax.dcg_metric, top_n=10),
3      "mrr@10": RaxMetric(rax.mrr_metric, top_n=10),
4  })
5
6  metrics.update(scores=scores, labels=labels, where=mask)
7  results = metrics.compute()
```

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

5 Numerical Stability

Optimizing complex likelihood expressions using gradient-based optimization 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 [19, 30]. Below, we cover the techniques CLAX uses to stabilize complex likelihood expressions by performing all probability computations in log-space.

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

$$\log\left(\prod_{i=1}^{n} p_i\right) = \sum_{i=1}^{n} \log p_i, \qquad (15)$$

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

Addition. While multiplication becomes more stable (and faster) in log-space, the addition of probabilities becomes more complicated as it requires first exponentiating log probabilities. This reintroduces the instabilities 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 [1]:

$$\texttt{log\_sum\_exp}(a) = a_{\max} + \log\left(\sum_{i=1}^{n} \exp(a_i - a_{\max})\right), \qquad (16)$$

where $a = (a_1, \ldots, a_n)$ is a vector of log values and $a_{\max} = \max_i(a_i)$ 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:

$$\log(\sigma(x)) = -\texttt{log\_sum\_exp}([0, -x]), \text{ or}$$
$$\log(1 - \sigma(x)) = -\texttt{log\_sum\_exp}([0, x]). \qquad (17)$$

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 [6]. 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 [19]. Therefore, we compute $\texttt{log1mexp}(x)$ as proposed in [38] and adopted by major frameworks such as TensorFlow[11]

and JAX.[12] Mächler [38] proposes a piecewise approximation that switches between two stable expressions that are precise in different input ranges.[13] For a log-probability $a \in \mathbb{R}, a \leq 0$:

$$\texttt{log1mexp}(a) = \begin{cases} \log(-\text{expm1}(a)) & \text{if } a > -\log(2) \\ \text{log1p}(-\exp(a)) & \text{if } a \leq -\log(2). \end{cases} \qquad (18)$$

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 A and their corresponding implementation can be found in our code repository.[14]

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.

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 [10, 47]. 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 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 [60]. 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 [8, 24, 35, 60, 61]. For ranking performance comparison, we use the subset of Baidu-ULTR created by Hager et al. [24] 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.

Implementation. All CLAX experiments use the default trainer with the AdamW optimizer [37] (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

---

[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]We direct interested readers for the theoretical motivation behind the switching point log(2) to [38, Section 2].

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

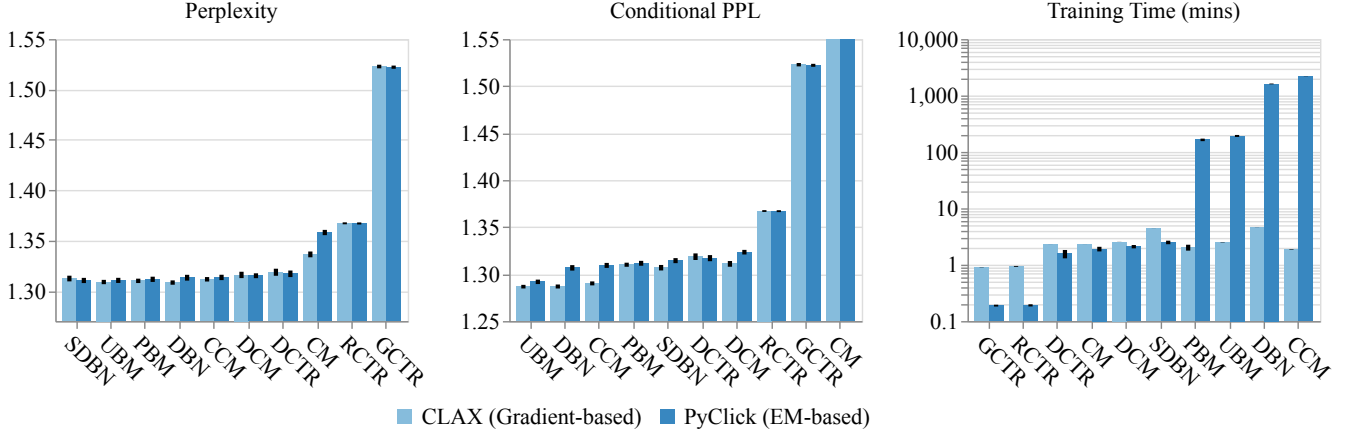[15]https://huggingface.co/datasets/philipphager/baidu-ultr_baidu-mlm-ctr

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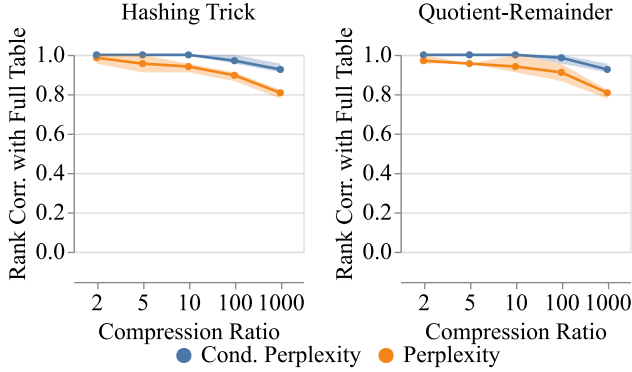

Figure 2: Kendall's τ between ranking models trained with and without embedding compression on WSCD-2012.

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 repository: https://github.com/philipphager/clax

7 Results

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 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.

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 2x to 1,000x.

Figure 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 10-100x. Unconditional perplexity is more susceptible to compression. However, as Figure 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 with 10x compression on three folds of the full Baidu-ULTR dataset containing over 1B user sessions. Fig. 3 shows the resulting models, all completing training under 2 hours.

Generalizing over features. Lastly, we parameterize CLAX's attractiveness and satisfaction parameters with a deep-cross network to

---

[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 [22].
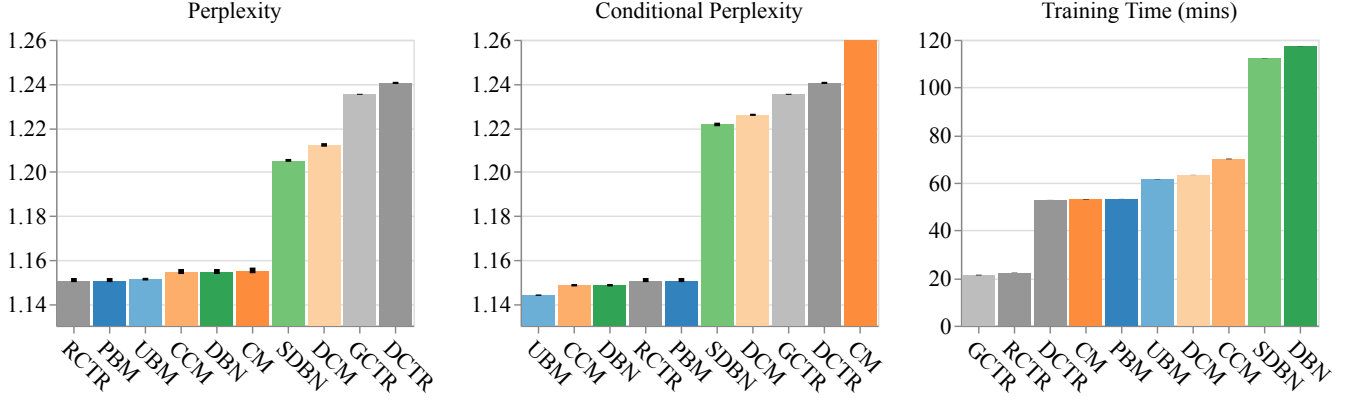
Figure 3: Embedding-based CLAX models on the Baidu-ULTR dataset (three folds of 800M / 200M / 200M sessions for training, validation, and testing) [60]. All models complete training under 2 hours using the hashing-trick with 10x compression.
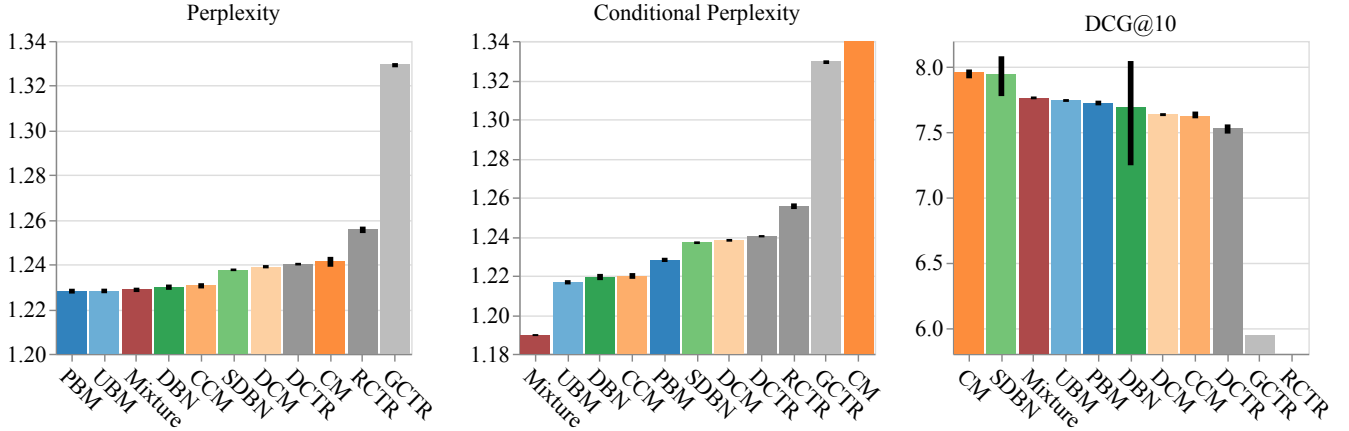

Figure 4: CLAX models generalizing over BERT features on the Baidu-ULTR-UVA dataset [24] using a deep-cross network achieve strong ranking performance and a different model fit compared to embedding-based models.

investigate click prediction and ranking performance when generalizing over query-document features. Figure 4 shows the results on the Baidu-ULTR-UVA subset [24]. 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 [24]. Nevertheless, cascade-based models achieve strong ranking performance, comparable to listwise LTR loss functions trained in previous work [24, Figure 3]. Our results suggest that complex click models can be effective ranking models.

Finally, we evaluate the effectiveness of our mixture model, which combines a PBM, DCTR, and GCTR model, in Figure 4, following the setup of Yan et al. [56] but excluding the RCTR as it cannot be applied to the Baidu-ULTR test. The mixture model achieves better model fit and ranking performance than each individual model.

8 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

References

[1] Pierre Blanchard, Desmond J. Higham, and Nicholas J. Higham. 2019. Accurate Computation of the Log-Sum-Exp and Softmax Functions. (2019). arXiv:1909.03469 [math.NA]

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

[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. 2018. *JAX: composable transformations of Python+NumPy programs.* http://github.com/jax-ml/jax

[4] Alberto Cabezas, Adrien Corenflos, Junpeng Lao, and Rémi Louf. 2024. BlackJAX: Composable Bayesian inference in JAX. arXiv:2402.10797 [cs.MS]

[5] Olivier Cappé. 2011. Online Expectation Maximisation. *Mixtures: Estimation and Applications* (2011), 31–53.

[6] Olivier Chapelle and Ya Zhang. 2009. A Dynamic Bayesian Network Click Model for Web Search Ranking. In *The World Wide Web Conference (WWW)*.

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

[8] 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 ACM International Conference on Web Search and Data Mining (WSDM)*.

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

[10] Aleksandr Chuklin, Ilya Markov, and Maarten de Rijke. 2015. *Click Models for Web Search.* Morgan & Claypool. https://doi.org/10.2200/S00654ED1V01Y201507ICR043

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

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

[13] 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.* http://github.com/google-deepmind

[14] 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).

[15] 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 (June 2024), 34 pages.

[16] 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. Series B (Methodological)* 39, 1 (1977), 1–38.

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

[18] Georges E. Dupret and Benjamin Piwowarski. 2008. A User Browsing Model to Predict Search Engine Click Data from Past Observations.. In *The International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*.

[19] David Goldberg. 1991. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys (CSUR)* 23, 1 (1991), 5–48.

[20] 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 *International Conference of the Cross-Language Evaluation Forum for European Languages (CLEF)*.

[21] 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 International Conference on World Wide Web (WWW)*.

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

[23] 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 ACM Conference on Recommender Systems (RecSys)*.

[24] 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 International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*.

[25] 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 *The ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (WSDM)*.

[26] 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

[27] Tom Hennigan, Trevor Cai, Tamara Norman, Lena Martens, and Igor Babuschkin. 2020. *Haiku: Sonnet for JAX.* http://github.com/deepmind/dm-haiku

[28] 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 ACM Conference on Recommender Systems (RecSys)*.

[29] 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 ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*.

[30] William Kahan. 1996. IEEE Standard 754 for Binary Floating-point Arithmetic. *Lecture Notes on the Status of IEEE* 754, 94720-1776 (1996), 11.

[31] 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 (WebConf)*.

[32] Patrick Kidger. 2021. *On Neural Differential Equations.* Ph.D. Dissertation. University of Oxford.

[33] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. arXiv:1412.6980 [cs.LG]

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

[35] 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 ACM International Conference on Web Search and Data Mining (WSDM)*.

[36] 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 International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*.

[37] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. (2019). arXiv:1711.05101 [cs.LG]

[38] Martin Mächler. 2012. Accurately Computing log (1- exp (-| a|)) Assessed by the Rmpfr package. *The Comprehensive R Archive Network* (2012), 1–9.

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

[40] 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 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*.

[41] 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.

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

[43] Jason Rader, Terry Lyons, and Patrick Kidger. 2023. Lineax: Unified Linear solves and Linear Least-Squares in JAX and Equinox. In *AI for science workshop at Neural Information Processing Systems 2023*.

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

[45] Sebastian Ruder. 2017. An overview of gradient descent optimization algorithms. arXiv:1609.04747 [cs.LG]

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

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

[48] 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 ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. 165–175.

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

[50] 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*.

[51] 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 ACM International Conference on Information and Knowledge Management (CIKM)*.

[52] 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 (WWW)*.

[53] 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 (WSDM)*.

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

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

[56] 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 International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*.

[57] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Anirudh 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 ACM Conference on Recommender Systems (RecSys)*.

[58] 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 ACM International Conference on Web Search and Data Mining (WSDM)*.

[59] 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 (WebConf)*.

[60] 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 *Advances in Neural Information Processing Systems (NeurIPS)*.

[61] 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

A Models

We introduce the ten click models implemented in CLAX following the standard work of Chuklin et al. [10, 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.

A.1 Global CTR model

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:

$$\log P(C = 1 \mid d, k) = \log \rho. \tag{19}$$

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:

$$\log P(C = 1 \mid d, k) = \log \theta_k. \tag{20}$$

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:

$$\log P(C = 1 \mid d, k) = \log \gamma_d. \tag{21}$$

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):

$$\log P(C = 1 \mid d, k) = \log \theta_k + \log \gamma_d. \tag{22}$$

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:

$$\log P(C = 1 \mid d, k) = \log \gamma_d + \sum_{i=1}^{k-1} \log(1 - \gamma_{d_i}). \tag{23}$$

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 [10]:

$$\log P(C = 1 \mid d, k, c_{<k}) = \begin{cases} \log \gamma_d & \text{if } \sum_{i=1}^{k-1} c_i = 0 \\ \text{min\_log\_prob} & \text{otherwise.} \end{cases} \tag{24}$$

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:

$$\log P(C = 1 \mid d, k, c_{<k}) = \log \theta_{k,k'} + \log \gamma_d, \tag{25}$$

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:

$$\log P(C = 1 \mid d, k) =$$
$$\log \left(\sum_{i=0}^{k-1} P(C = 1 \mid d_i, i) \cdot \left(\prod_{j=i+1}^{k-1} (1 - \theta_{j,i} \gamma_{d_j}) \right) \theta_{k,i} \gamma_d \right). \tag{26}$$

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.

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:

$$\log P(C = 1 \mid d, k) = \log(\epsilon_k) + \log(\gamma_d)$$
$$\log(\epsilon_{k+1}) = \log(\epsilon_k) + \log(\gamma_{d_k}\lambda_k + (1 - \gamma_{d_k})). \quad (27)$$

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

$$\log P(C = 1 \mid d, k, c_{<k}) = \log(\epsilon_k) + \log(\gamma_d)$$
$$\log(\epsilon_{k+1}) = \log\left(c_k\lambda_k + (1 - c_k)\frac{(1 - \gamma_{d_k})\epsilon_k}{1 - \gamma_{d_k}\epsilon_k}\right). \quad (28)$$

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.

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:

$$\log P(C = 1 \mid d, k) = \log(\gamma_d) + \log(\epsilon_k)$$
$$\log(\epsilon_{k+1}) = \log(\epsilon_k)$$
$$+ \log\left(\gamma_{d_k}((1 - \gamma_{d_k})\tau_2 + \gamma_{d_k}\tau_3)\right) \quad (29)$$
$$+ (1 - \gamma_{d_k})\tau_1\right).$$

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:

$$\log P(C = 1 \mid d, k, c_{<k}) = \log(\gamma_d) + \log(\epsilon_k)$$
$$\log(\epsilon_{k+1}) = c_k\left[\log\left(\gamma_{d_k}\tau_3 + (1 - \gamma_{d_k})\tau_2\right)\right]$$
$$+ (1 - c_k)\left[\log(1 - \gamma_{d_k}) + \log(\epsilon_k)\right. \quad (30)$$
$$+ \log(\tau_1) - \log(1 - \gamma_{d_k}\epsilon_k)\right].$$

A.9 Dynamic Bayesian network

The dynamic Bayesian network (DBN) model 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:

$$\log P(C = 1 \mid d, k) = \log(\gamma_d) + \log(\epsilon_k)$$
$$\log(\epsilon_{k+1}) = \log(\epsilon_k) + \log(\lambda) + \log(1 - \gamma_{d_k}\sigma_{d_k}). \quad (31)$$

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:

$$\log P(C = 1 \mid d, k, c_{<k}) = \log(\gamma_d) + \log(\epsilon_k)$$
$$\log(\epsilon_{k+1}) = \log(\lambda) + c_k\left[\log(1 - \sigma_{d_k})\right]$$
$$+ (1 - c_k)\left[\log(1 - \gamma_{d_k}) + \log(\epsilon_k)\right. \quad (32)$$
$$- \log(1 - \gamma_{d_k}\epsilon_k)\right].$$