Learning to Rank Everything, Everywhere and All at Once

A Perspective on the Future of Search and Recommender System Optimization

Harrie Oosterhuis
University of Amsterdam



Universiteit van Amsterdam
Instituut voor Informatica

About The Speaker

Dr. Harrie Oosterhuis



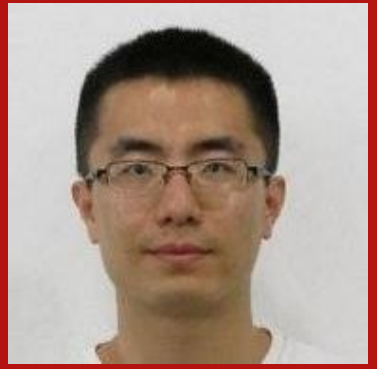
- Associate Professor at the University of Amsterdam
  - Machine Learning for Information Retrieval and RecSys.
  - Focus on optimization for ranking (from user interactions).

- Part-time visiting researcher in industry:
  - 2026-now - **Amazon Music** - *Off-policy evaluation for RecSys.*
  - 2023-2025 - **Google DeepMind** - *Large language models (LLMs) for ranking.*
  - 2022 - **Twitter** - *Learning to rank for RecSys.*

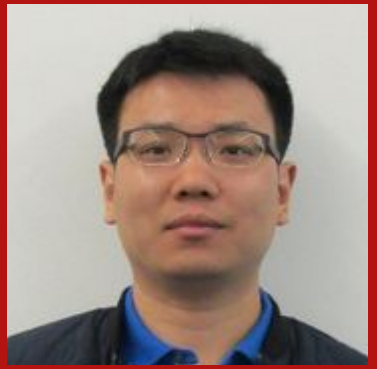
Thanks to my Collaborators!



Rolf Jagerman
Google Deepmind



Zhen Qin
Spotify



Xuanhui Wang
Google Deepmind

The Movie vs. The Talk



This Talk

Description:

- chaotic, maximalist sci-fi action
- deeply moving story about family and existential meaning
- emotional depth & hilarious comedy

- equations, algorithms and graphs
- profound, touching perspective on ranking model optimization
- very funny for a research talk (1 or 2 chuckles)

Budget:	$14-25 million	$0-0.1 million
Admission:	$11	$400-$1450

Learning to Rank Everything, Everywhere and All at Once

Part 1: Every Objective Function

Go beyond relevance metrics.

Simple implementation, high efficiency and effectiveness.

Part 2: Every Ranking System with Joint Optimization

Ranking systems are not monolithic models.

Optimize the design of the system; open your mind to new designs.

Part 0:

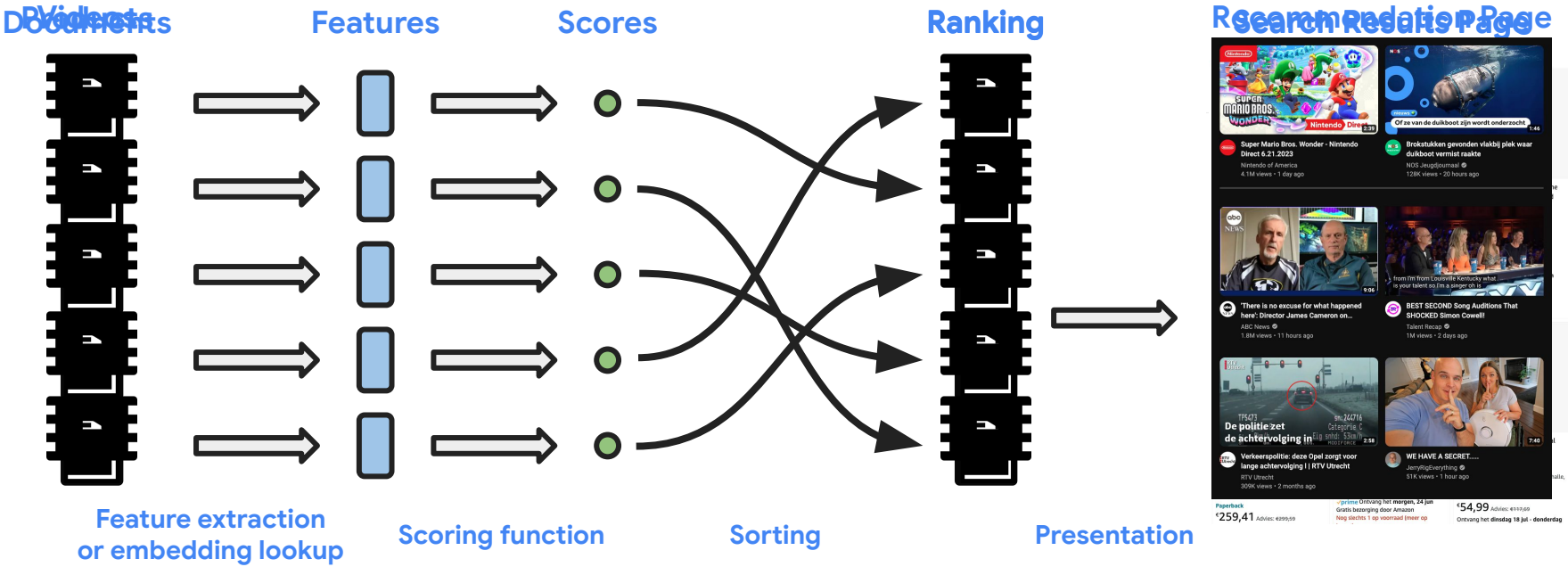
Part 0: **Learning to Rank**



Part 0: Learning to Rank

Introduction

Introduction: Ranking for Search and Recommendation



Ranking and Learning to Rank

Ranking is a **common denominator** of **search and recommendation**.

For this talk, **ranking** is defined as:

- Score per item, followed by a sorting operation (scores can be stochastic).
- Items are **scored independently**, there is **no listwise** construction.

(Top-1 ranking is not a real ranking problem.)

For this talk, I assume that **item relevance / quality** is **known**:

- Learning from user interactions is important but not discussed today.

Part 1:

Part 1: **Everything**



Part 1: Everything

Exposure-Based Reinforcement Learning to Rank

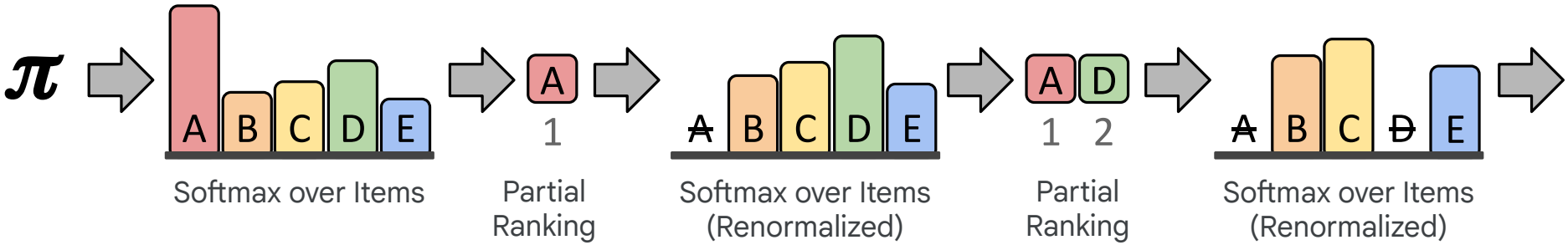
Harrie Oosterhuis, Rolf Jagerman, Zhen Qin and Xuanhui Wang

Based on the full paper with the same title, accepted at the ACM ICTIR 2026 conference.

Reinforcement Learning to Rank

Ranking is treated as a **decision problem** made by a **stochastic policy**:

- i.e., a **probabilistic distribution** over rankings,
- a **Plackett-Luce** ranking model: **sampling** without replacement from a **softmax**.



Sampling is very **efficient**: add **Gumbel noise** to scores and sort (once).

Positioning and Contributions

Previous work in Reinforcement Learning to Rank

- Specialized **custom gradient implementations** for **fast computation**.
- Designed for **CPU** and only tested for a few epochs (<300).
- Does not work well with **GPU** or **auto-differentiation**.

Our Approach

- Design for **GPU computation**.
- Avoid custom gradients to work with **auto-differentiation**.
- Focus on applying **established techniques** from reinforcement learning.

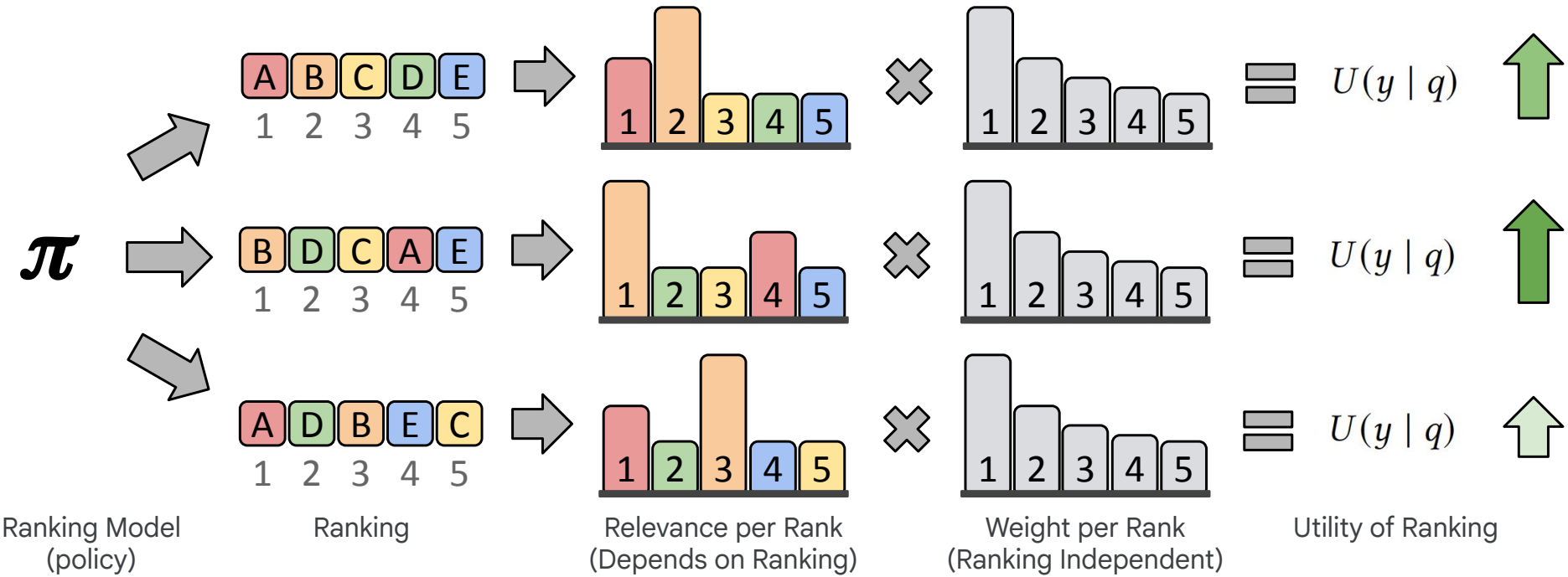
Our Contributions

- Adding **baseline corrections** to gradient estimators.
- A novel **ranking marginalization** approach for higher sample efficiency.
- A novel **exposure-based abstraction** for **simplifying implementation.**

Method

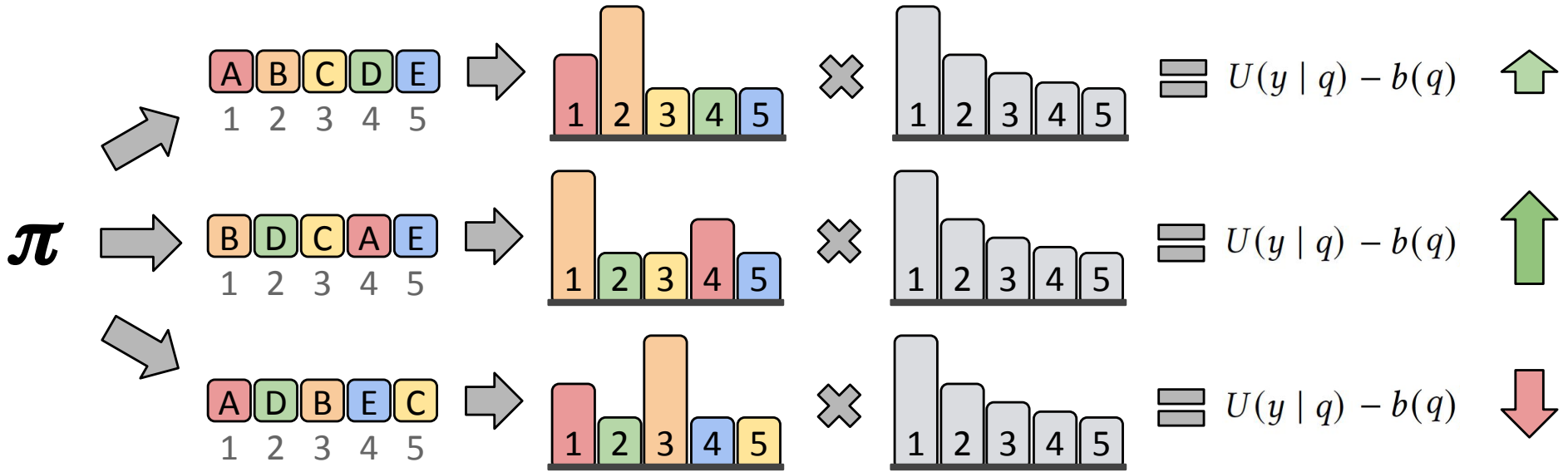
Reinforcement Learning to Rank

Optimize a probabilistic ranking model based on sampled rankings.



Baseline Corrections

Optimize the difference of utility with the mean utility for more efficient learning.



A Novel Marginalization Strategy

Standard: The sampled ranking is a single action.



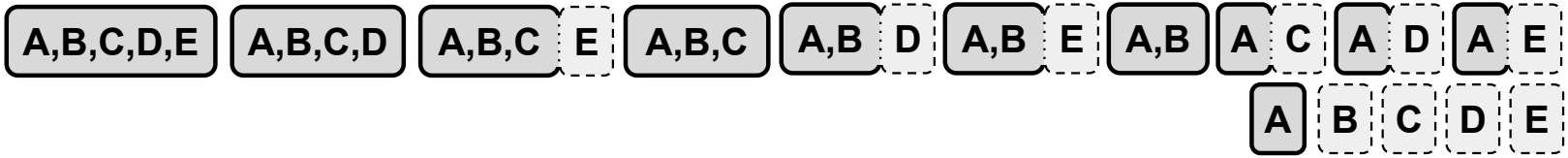
Placement: Every document placement in the sampled ranking is an action.



Marginalize First: Consider every possible counterfactual first placement.



Marginalize All Positions: Consider every possible counterfactual placement at each position.



A Novel Marginalization Strategy for Learning to Rank

Standard policy-gradient treats an **entire ranking as one action**:

$$y^{(i)} \sim \pi(q) \qquad U(\pi \mid q) \approx \frac{1}{N} \sum_{i=1}^{N} U(y^{(i)} \mid q)$$

Per-placement policy-gradient treats the **each sub-ranking as an action**:

$$U(\pi \mid q) = \sum_{k=1}^{K} \theta_k \sum_{y \in \pi(q)} \pi(y \mid q) \rho_{y_k} = \sum_{k=1}^{K} \theta_k \sum_{y_{1:k} \in \pi(q)} \pi(y_{1:k} \mid q) \rho_{y_k}$$

We can **marginalize** over the **first** document **placement**:

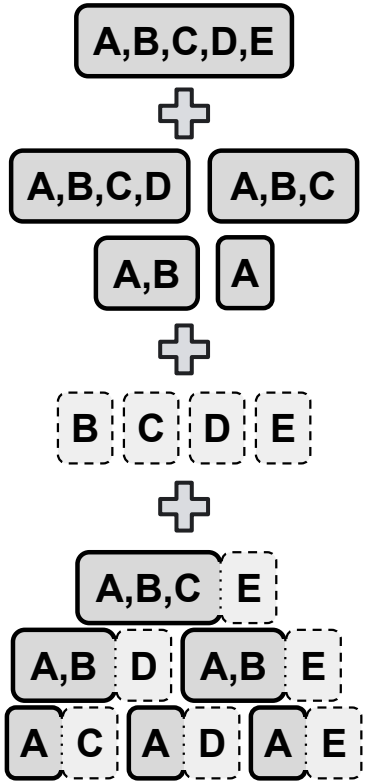
$$U(\pi \mid q) \approx \theta_1 \left(\sum_{d \in D} \pi(d \mid \emptyset) \rho_d \right) + \frac{1}{N} \sum_{i=1}^{N} \sum_{k=2}^{K} \theta_k \rho_{y_k^{(i)}}$$

even **marginalize** over **every placement** while keeping the **sampled prefix**:

$$U(\pi \mid q) \approx \sum_{k=1}^{K} \theta_k \frac{1}{N} \sum_{i=1}^{N} \sum_{d \in D} \pi\left(d \mid y_{1:k-1}^{(i)}\right) \rho_d = \frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} \theta_k \sum_{d \in D} \pi\left(d \mid y_{1:k-1}^{(i)}\right) \rho_d$$

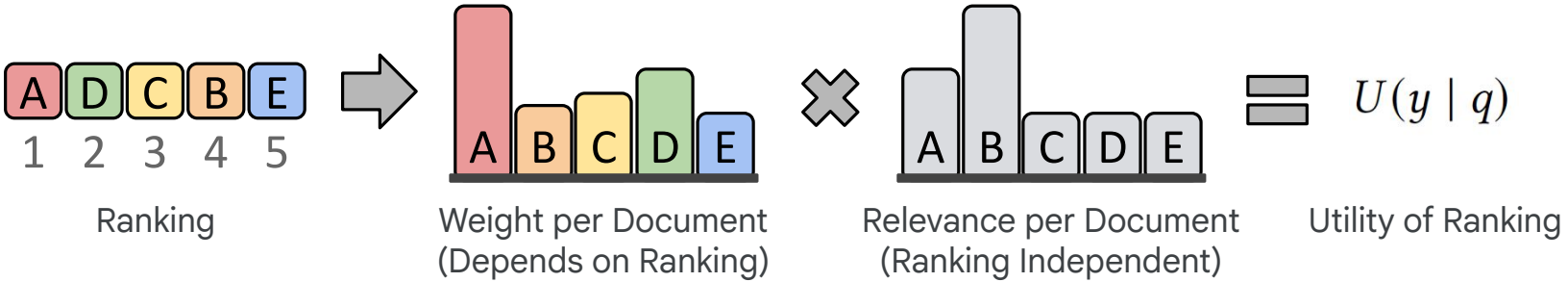
For *N* samples, our policy-gradient is then based on this many sub-rankings:

$$N(D + D - 1 + D - 2 + \ldots + D - (K - 1)) = N\left(DK - \binom{K}{2}\right)$$



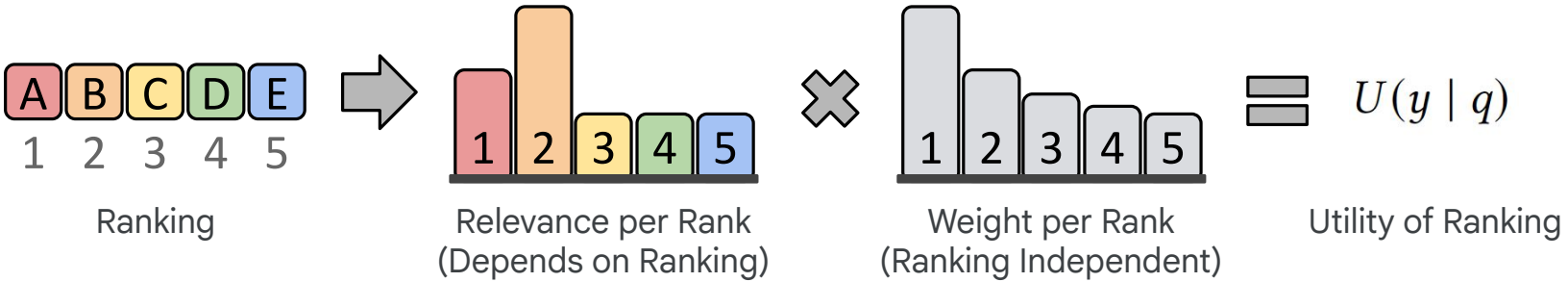
Exposure-based Framing on Rankings

Frame ranking as the **allocation of exposure** (rank weights) to documents.



Ranking | Weight per Document (Depends on Ranking) | Relevance per Document (Ranking Independent) | Utility of Ranking

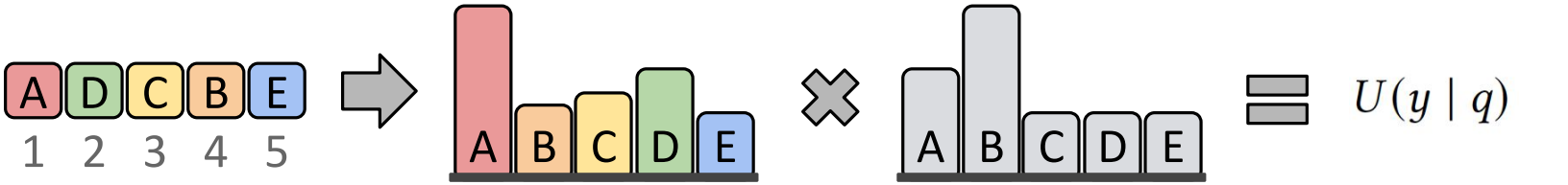
For comparison, this was the previous framing:



Ranking | Relevance per Rank (Depends on Ranking) | Weight per Rank (Ranking Independent) | Utility of Ranking

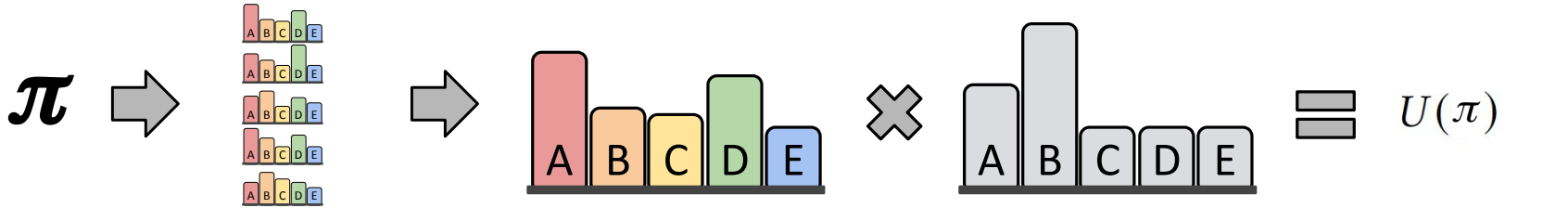
Exposure-based Framing on Rankings

Frame ranking as the **allocation of exposure** (rank weights) to documents.



Ranking | Weight per Document (Depends on Ranking) | Relevance per Document (Ranking Independent) | Utility of Ranking

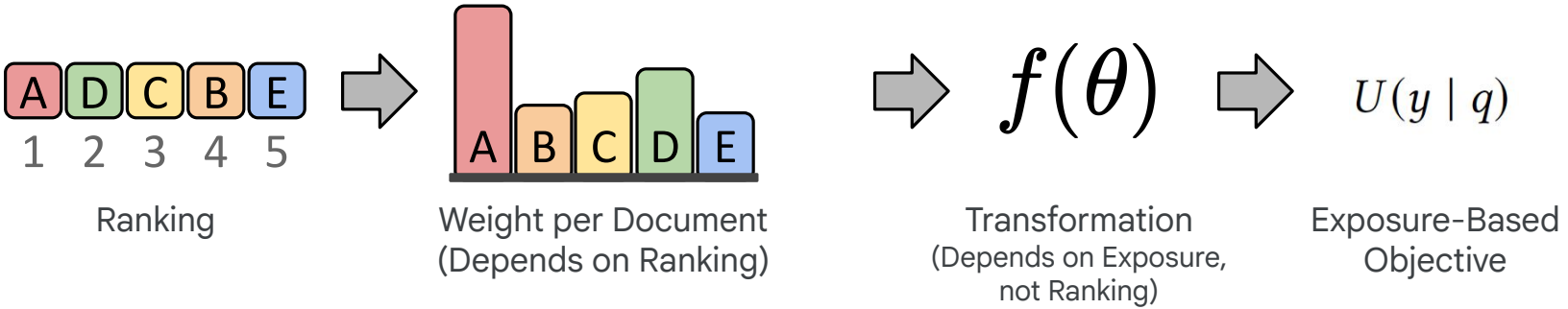
The **exposure distribution** can also be done on **policy-level**:



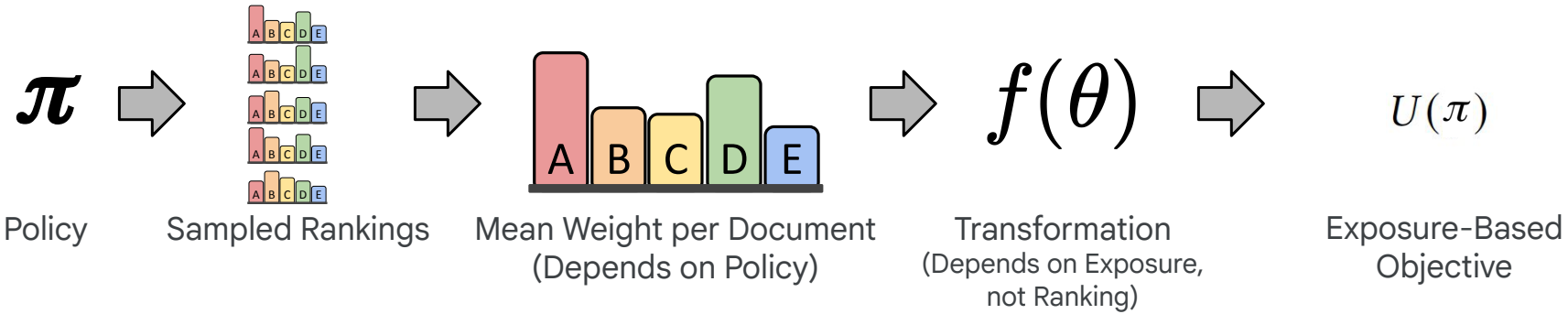
Policy | Sampled Rankings | Mean Weight per Document (Depends on Policy) | Relevance per Document (Policy Independent) | Utility of Policy

Exposure-based Objectives

Many objectives are functions over the **exposure distribution** (fairness, distillation, etc.):



On **policy-level**:



Exposure-based Learning to Rank

Instead of utility, we propose an **estimator** for **document exposure** (using marginalization):

$$\theta_{d|\pi,q} = \mathbb{E}_{y\sim\pi(q)}\left[\theta_{\text{rank}(d|y)}\right] \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\pi\left(d \mid y^{(i)}_{1:k-1}\right)\theta_k$$

Resulting in the following policy-gradient:

$$\frac{\delta}{\delta f}\theta_{d|\pi,q} \approx \frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{K}\left(\theta_k - b^{(i)}_d\right)\left(\left[\frac{\delta}{\delta f}\pi\left(d \mid y^{(i)}_{1:k-1}\right)\right] + \pi\left(d \mid y^{(i)}_{1:k-1}\right)\left[\frac{\delta}{\delta f}\log\pi\left(y^{(i)}_{1:k-1} \mid q\right)\right]\right)$$

$$- b^{(i)}_d \mathbb{1}\left[d \notin y^{(i)}_{1:K-1}\right]\left(\left[\frac{\delta}{\delta f}\left(1 - \pi\left(d \mid y^{(i)}_{1:K-1}\right)\right)\right] + \left(1 - \pi\left(d \mid y^{(i)}_{1:K-1}\right)\right)\left[\frac{\delta}{\delta f}\log\pi\left(y^{(i)}_{1:K-1} \mid q\right)\right]\right),$$

$$b^{(i)}_d = \frac{1}{N-1}\sum_{j=1}^{N}\mathbb{1}[i \neq j]\sum_{k=1}^{K}\pi\left(d \mid y^{(j)}_{1:k-1}\right)\theta_k$$

Due to the **multivariable chain rule**, we can optimize **any exposure-based loss** function:

$$\frac{\delta}{\delta f}L(\boldsymbol{\theta}_{\pi|q}) = \sum_{d\in D}\left[\frac{\delta L(\boldsymbol{\theta}_{\pi|q})}{\delta\theta_{d|\pi,q}}\right]\left[\frac{\delta\theta_{d|\pi,q}}{\delta f}\right]$$

Implementation

Exposure Estimator Implementation

The exposure-based abstraction **greatly simplifies the implementation of new loss functions**.

Plug-and-play usage due to **multivariable chain rule**.

```python
import jax.numpy as jnp
from jax.lax import stop_gradient
from utils import sample_rankings, cumlogsumexp
# input: [D] vector of scores, [K] vector of exposure, RNG key and N
# output: [D] vector of (estimated) expected exposure per document
def exposure(scores, rank_exposure, rng_key, n_samples):
  K = rank_exposure.shape[0]
  # creates [N, D] matrix of sampled rankings from PL ranking model
  rankings = sample_rankings(rng_key, n_samples, scores)
  ranked_scores = scores[rankings]
  # [N, K] log denominator of placement prob. per position per sample
  log_denom = cumlogsumexp(ranked_scores, axis=1, reverse=True)[:,:K]
  # [N, K-1] log prob. of sampled ranking up to pos. k (not including)
  log_prefix = ranked_scores[:,:K-1] - log_denom[:,:K-1]
  log_prefix = jnp.cumsum(jnp.pad(log_prefix, ((0,0),(1,0))), axis=1)
  # [N, K, D] placement prob. of every document per position & sample
  prob_in = jnp.exp(scores[None,None,:] - log_denom[:,:,None])
  mask = jnp.zeros_like(prob_in, dtype=bool)
  mask = mask.at[jnp.arange(n_samples)[:,None], jnp.arange(1,K)[None,:],
                 rankings[:,:K-1]].set(True)
  mask = jnp.cumsum(mask, axis=1).astype(bool)
  prob_in = jnp.where(mask, 0, prob_in)
  prob_out = jnp.where(mask[:,-1,:], 0, 1-prob_in[:,-1,:])
  # [N, K, D] placement prob. per position and outside rankings
  prob_all = jnp.concat((prob_in, prob_out[:,None,:]), axis=1)
  log_prefix_all = jnp.concat((log_prefix, log_prefix[:,-1:]), axis=1)
  rank_exposure_all = jnp.pad(rank_exposure, (0,1))
  # [N, K, D] loss value used for computing policy gradient
  loss_K_D = prob_all + stop_gradient(prob_all)*log_prefix_all[:,:,None]
  exposure_prob_prod = prob_all * rank_exposure_all[None,:,None]
  # [N, D] exposure of document per sample
  sample_exposure = jnp.sum(exposure_prob_prod, axis=1)
  # [D] expected exposure of document estimated from samples
  mean_exposure = jnp.mean(sample_exposure, axis=0)
  # [N, D] leave-one-out baseline corrections
  baseline = mean_exposure[None,:] - sample_exposure/n_samples
  baseline *= n_samples / (n_samples-1)
  # [N, K, D] baseline-corrected rewards for every doc., pos., sample
  rewards = stop_gradient(rank_exposure_all[None,:,None]-baseline[:,None,:])
  loss = jnp.mean(jnp.sum(rewards * loss_K_D, axis=1), axis=0)
  # trick: value is expected exposure but gradient comes from loss
  return stop_gradient(mean_exposure) + loss - stop_gradient(loss)
```

Listing 3: Implementation of the exposure-based estimator.

Implementing New Loss Functions

Three loss functions implemented in **8 lines** of code.

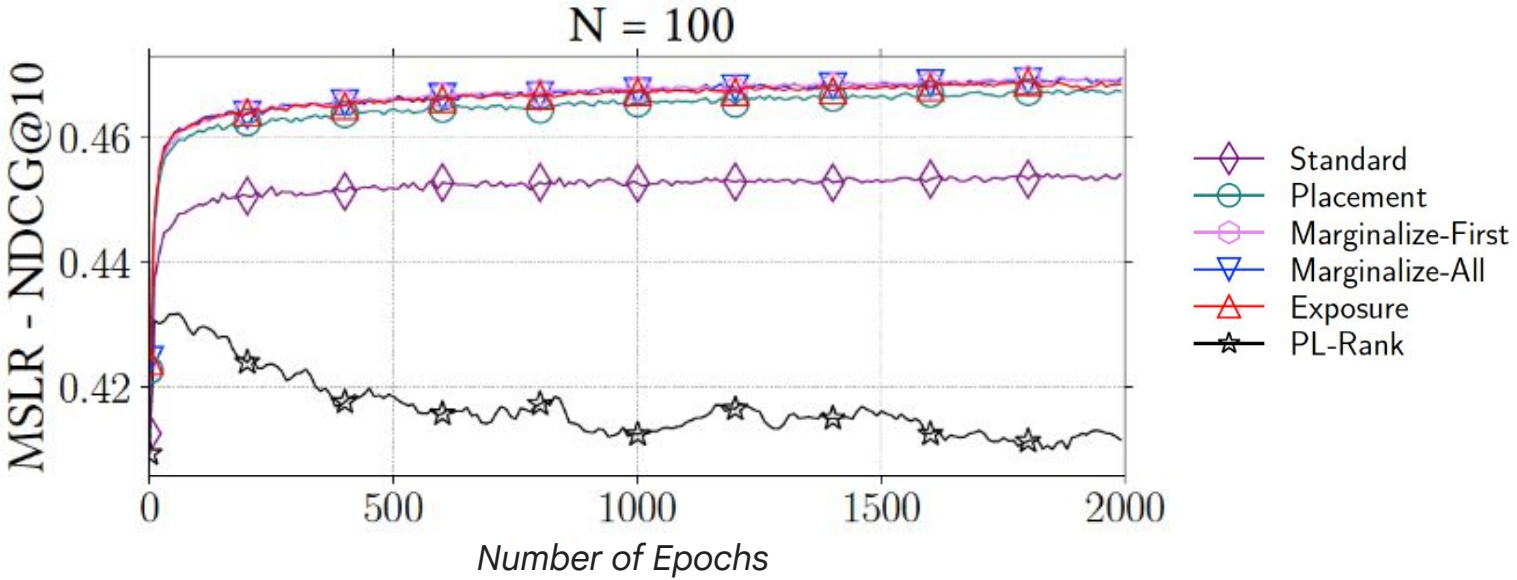
```
exposures = exposure(scores, rank_exposure, rng_key, n_samples)
# relevances is a [D] vector of relevance labels per doc.
relevance_loss = -jnp.sum(exposures * relevances)

ratios = exposures / relevances
frac_fair_loss = jnp.mean((ratios[:,None] - ratios[None,:])**2)
frac_fair_loss *= ratios.size**2 / (ratios.size**2 - ratios.size)

norm_exp = exposures / jnp.sum(exposures)
norm_rel = relevances / jnp.sum(relevances)
KL_fair_loss = -jnp.sum(norm_rel * jnp.log(norm_exp / norm_rel))
```

Experiments

Experimental Results - Without Baseline Corrections



- **PL-Rank** is **highly unstable** and **cannot converge** reliably.
  - Previous work only tested 40 or ~250 epochs with 64 bit floats. We used 32 bit floats.
- **Marginalization** greatly **increases learning speed** and seemingly the **point of convergence**.

Experimental Results - Without Baseline Corrections

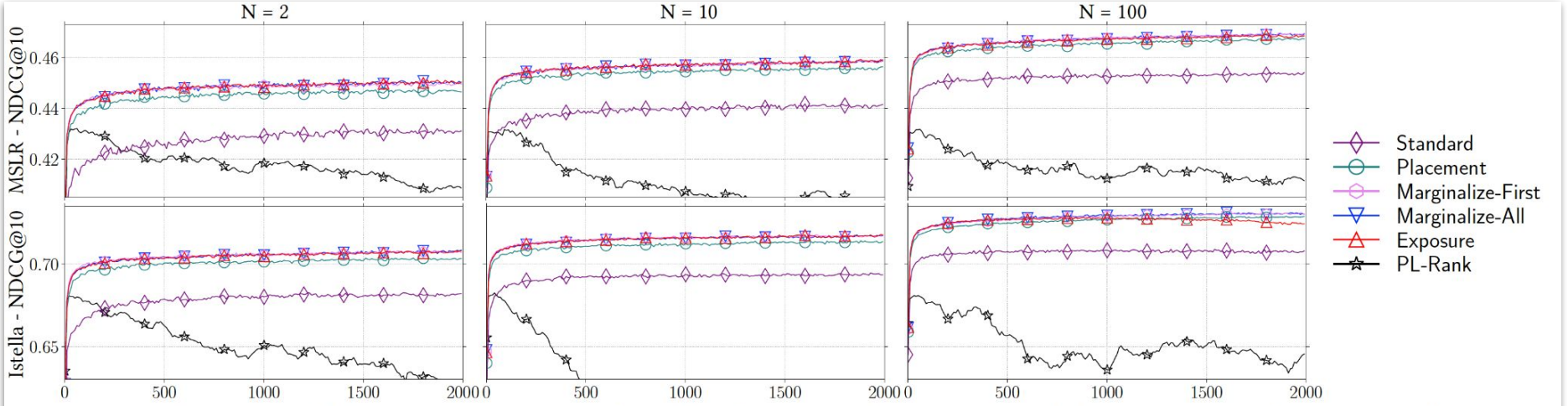
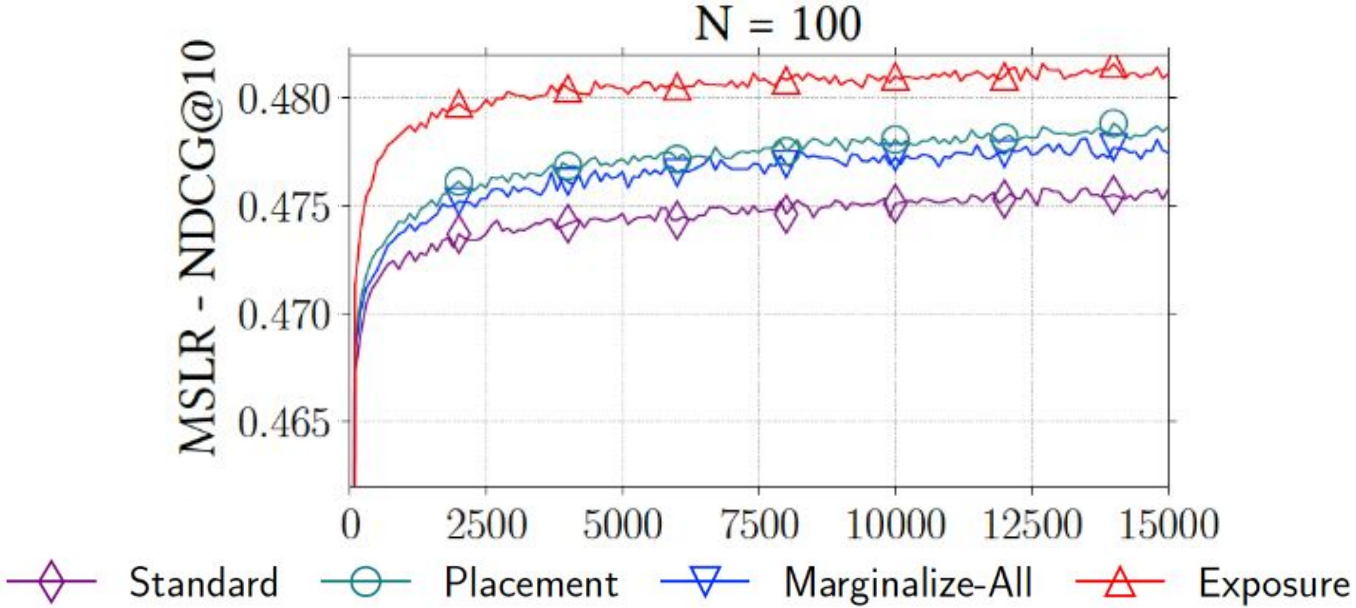


Figure 2: Performance of different estimators *without* baseline corrections (i.e., $b = 0$) for varying numbers of samples (N) per gradient estimate. Y-axis: NDCG@10 on test set; X-axis: training epochs. Top-row: MSLR-Web30k; Bottom-row: Istella-S.

- **PL-Rank** is **highly unstable** and **cannot converge** reliably.
  - Previous work only tested 40 or ~250 epochs with 64 bit floats. We used 32 bit floats.
- **Marginalization** greatly **increases learning speed** and seemingly the **point of convergence**.

Experimental Results - With Baseline Corrections



- **Baselines** have an enormously **greater impact** than **marginalization**.
- **Marginalization** performs substantially worse than other methods when N < 100.
  - We think due to the mismatch with baseline estimate.
- **Exposure-based** greatly **increases learning speed** and possibly the **point of convergence**.
  - As a result, overfitting starts earlier on Istella-S.

Experimental Results - With Baseline Corrections

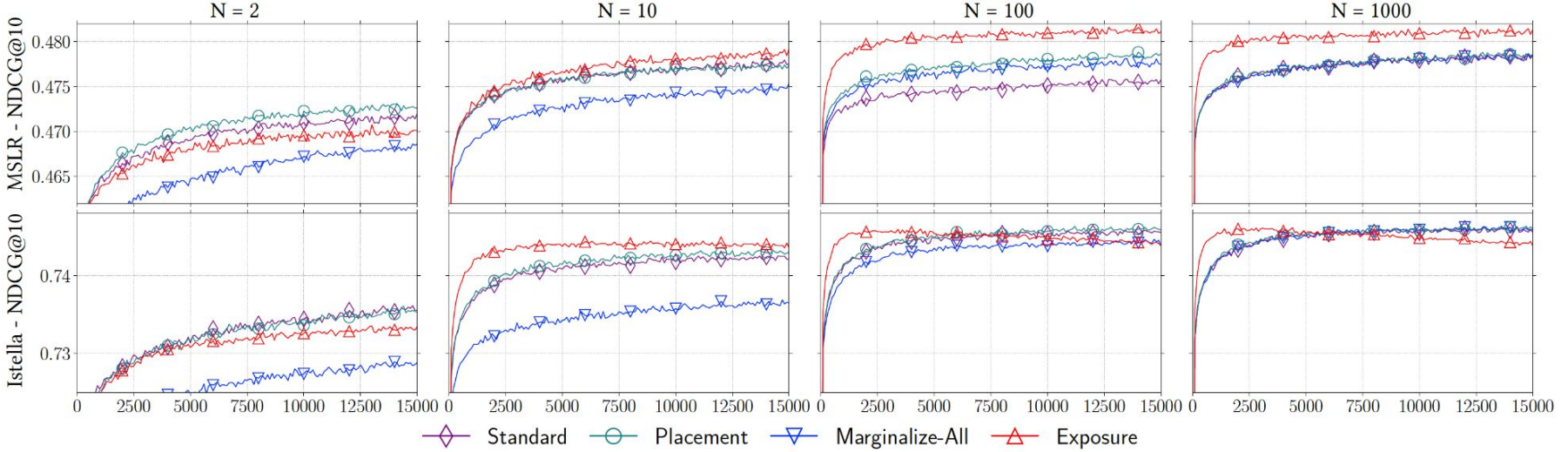


Figure 3: Performance of different estimators *with* baseline corrections for varying numbers of samples (N) per estimate. Y-axis: NDCG@10 on test set; X-axis: training epochs. Top-row: MSLR-Web30k; Bottom-row: Istella-S.

- **Baselines** have an enormously **greater impact** than **marginalization**.
- **Marginalization** performs substantially worse than other methods when N < 100.
  - We think due to the mismatch with baseline estimate.
- **Exposure-based** greatly **increases learning speed** and possibly the **point of convergence**.
  - As a result, overfitting starts earlier on Istella-S.

Experimental Results - Time per Epoch

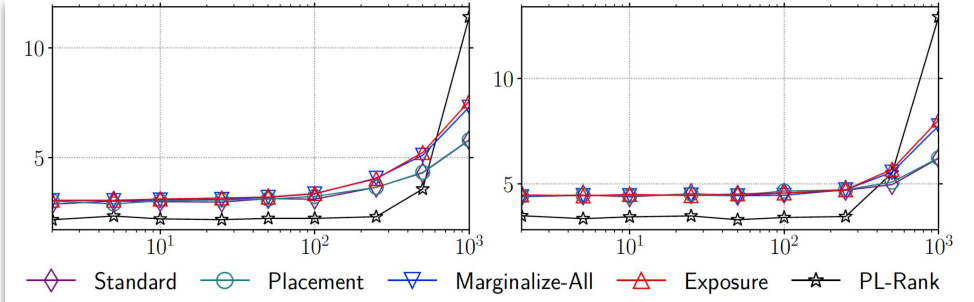


Figure 4: Seconds (Y-axis) used per training epoch by different estimators for varying numbers of samples N (X-axis). Left: MSLR-Web30k; Right: Istella-S LETOR.

Time measured on JAX implementation, each run was performed on a single GPU:

- NVIDIA® Tesla® V100 for NVLink with 64GiB RAM and 32 bit floats.

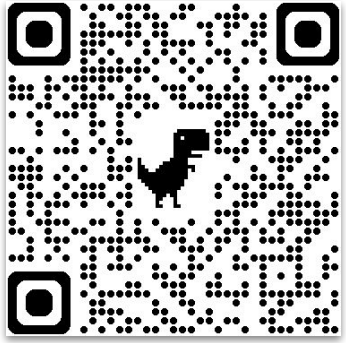
Observations:

- Our methods are slower than PL-Rank, but that was highly unstable.
- When $N \leq 200$, all our methods have comparable runtimes. ($N = 100$ is enough for effectiveness)
  - **No meaningful penalty for marginalization or exposure-based estimation.**

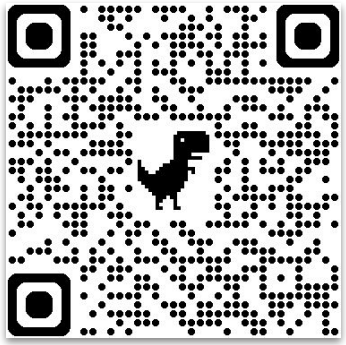
Conclusion

Contributions

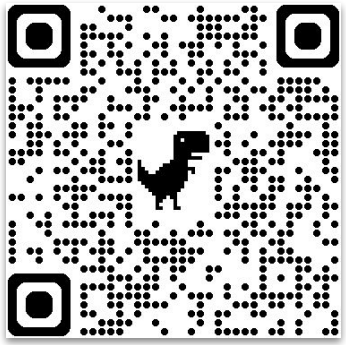
- **Marginalization** approach for reinforcement learning to rank:
  - Lower variance gradient estimation.
  - No meaningful added computational costs on GPU.
- **Exposure-based learning to rank**:
  - Greatly **simplifies implementation** of new loss functions.
  - Enormous **gain** in **learning speed**.
- Found significant **instability** in **PL-Rank.**
- **Reinforcement LTR is better and easier to work with.**
- **We can optimize every exposure-based metric!**



https://doi.org/10.1145/3805713.3820396



https://github.com/google/rax



https://github.com/HarrieO/2026-ictir-exposure-LTR

Part 2:

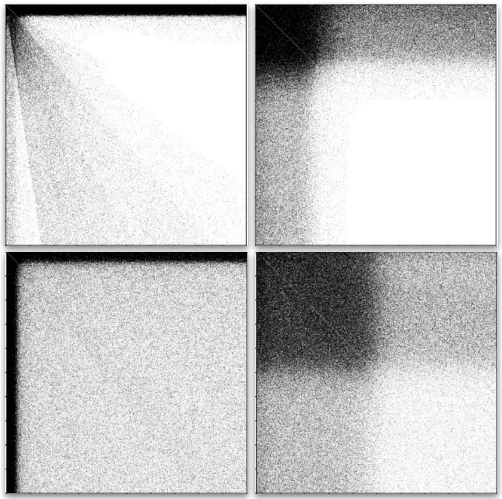
Part 2: **Everywhere and All at Once**



Part 2: Everywhere and All at Once

Optimizing Compound Retrieval Systems

Harrie Oosterhuis, Rolf Jagerman, Zhen Qin and Xuanhui Wang



Based on the full paper with the same title, accepted at the ACM SIGIR 2025 conference.

Introduction

Modern retrieval systems use **multiple prediction models** to form rankings in response to queries.

Thus, an important question for the field is:

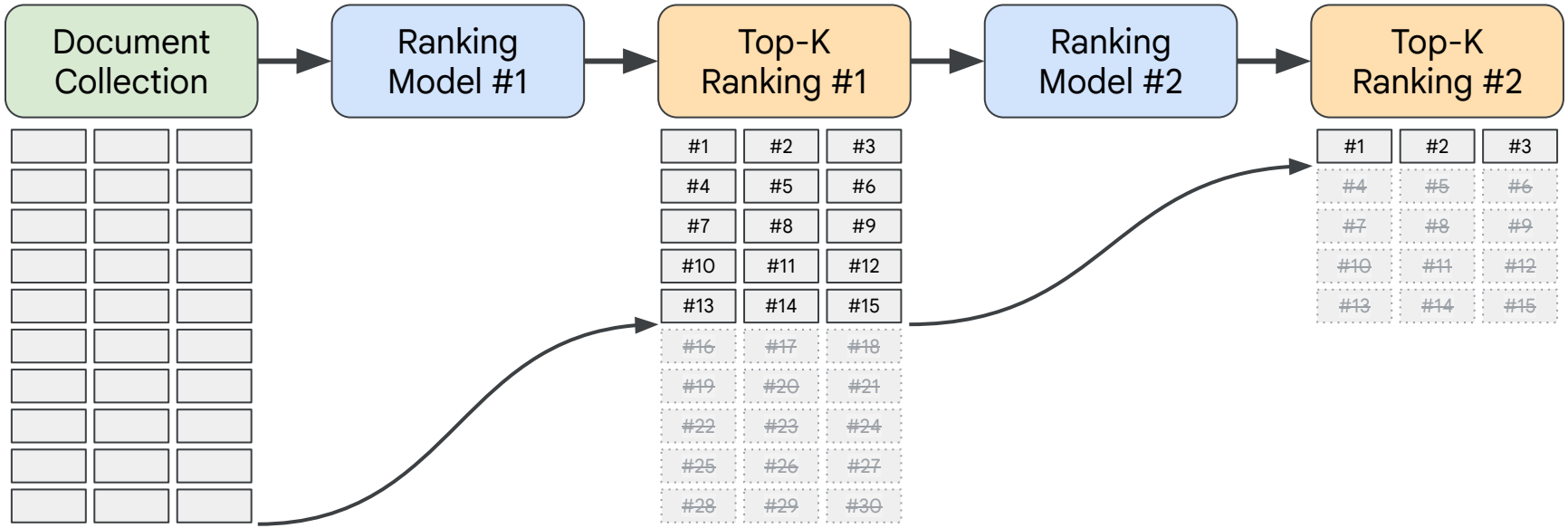
How should retrieval systems utilize multiple prediction models to form rankings in the most effective and efficient manner?

Consensus answer in the information retrieval field:

with **Cascading Retrieval Systems.**

Background: Cascading Retrieval Systems

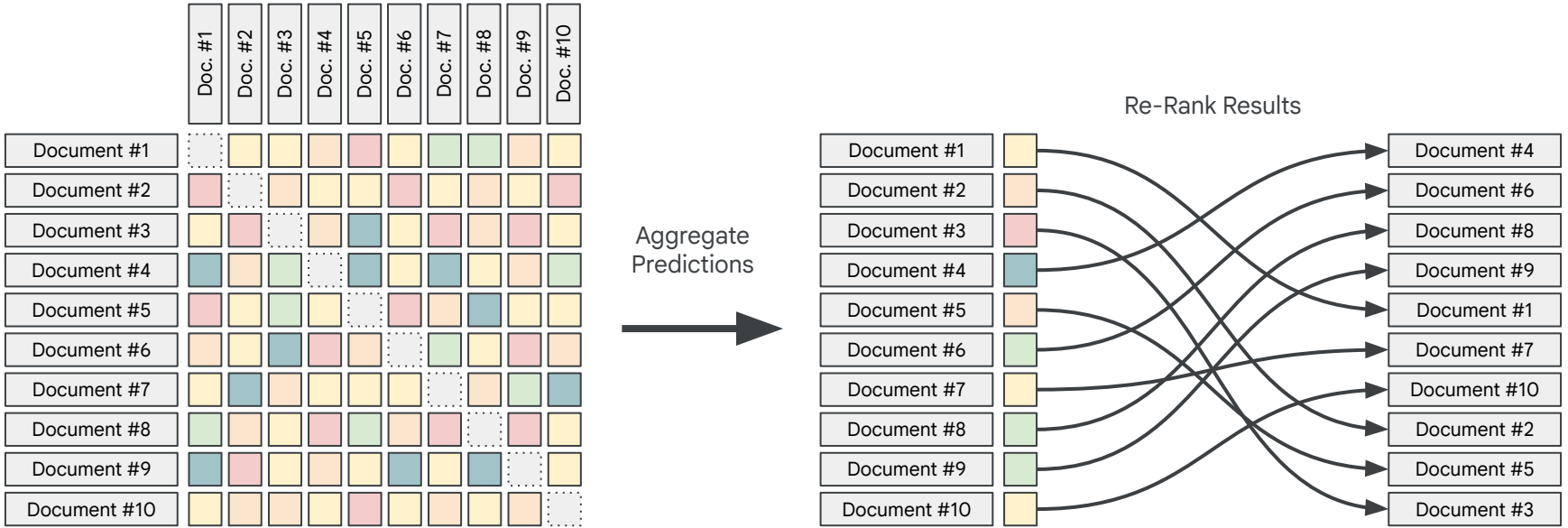
- Each model re-ranks a top-K of the previous model.
- In order of model cost, cheapest model ranks the most documents, costliest the fewest.
- Better efficiency-effectiveness trade-off than any individual model.



Lidan Wang, Jimmy Lin, and Donald Metzler. 2011. A cascade ranking model for efficient ranked retrieval. In Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval. 105–114.

Background: Pairwise Relevance Prompting (PRP)

- Ask an LLM to predict whether one document is more relevant than another. (Normalize logit scores.)
- Aggregate predictions across all pairs and re-rank accordingly.
- Quadratic scaling makes it impractical, even when parallelized and applied in a cascading system.



Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, et al. 2024. Large Language Models are Effective Text Rankers with Pairwise Ranking Prompting. In Findings of the Association for Computational Linguistics: NAACL 2024. 1504–1518.

Motivation

Our main question:

How should retrieval systems utilize multiple prediction models to form rankings in the most effective and efficient manner?

Consensus answer in the information retrieval field:

with **Cascading Retrieval Systems.**

However there are shortcomings:

- Pairwise Relevance Prompting (PRP) does not scale in the cascading paradigm.

And it appears that **alternatives** to the cascading paradigm **have never been explored**.

Compound Retrieval Systems

Novel concept defined as:

> **Any retrieval system that constructs a ranking by using multiple prediction models.**

Intentionally broad and a generalization of cascading retrieval systems.

Designs depend on two essential questions:

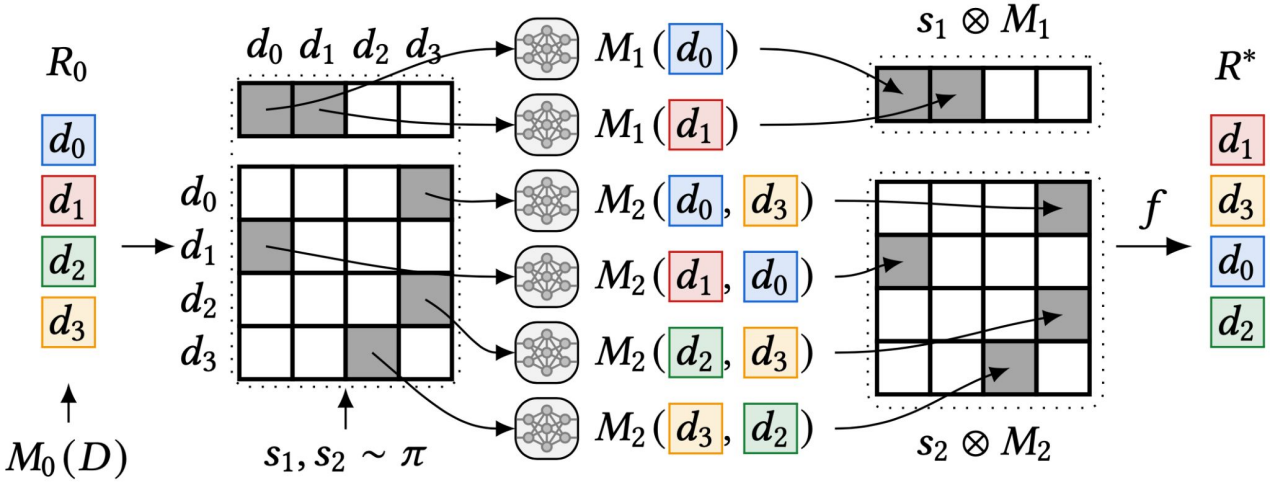
- **What predictions should the component models make?**
- **How should predictions be combined to construct a ranking?**

Start of a novel research direction to explore compound retrieval system designs and optimization.

Our Optimized Compound Retrieval System

- Q1: *What predictions should the component models make?*
  - A selection policy π that decides what predictions to gather.
- Q2: *How should predictions be combined to construct a ranking?*
  - A differentiable function f that transforms sets of predictions into ranking scores.

For example, for a first stage ranker M_0; a pointwise predictor M_1 and a pairwise predictor M_2:



End-to-end Gradient Estimation

Our goal is a linear tradeoff between ranking performance and computational costs:

$$\mathcal{L}_{\text{comp}}(f, \pi) = \alpha \mathcal{L}_{\text{ranking}}(\pi, f) + (1 - \alpha)\mathcal{L}_{\text{cost}}(\pi)$$

The aggregation function f sums each selected prediction after a linear transformation:

- A score A_r is given for each rank in the first stage-ranking: $f_0(r) = A_r$
- Pointwise or pairwise predictions are linearly transformed with intercept B_r and slope C_r:
  (Zero when prediction is not selected.)

$$f_1(s, m, r) = s \cdot (B_r + C_r \cdot m)$$
$$f_2(s, m, r_1, r_2) = s \cdot (B_{r_1,r_2} + C_{r_1,r_2} \cdot m)$$

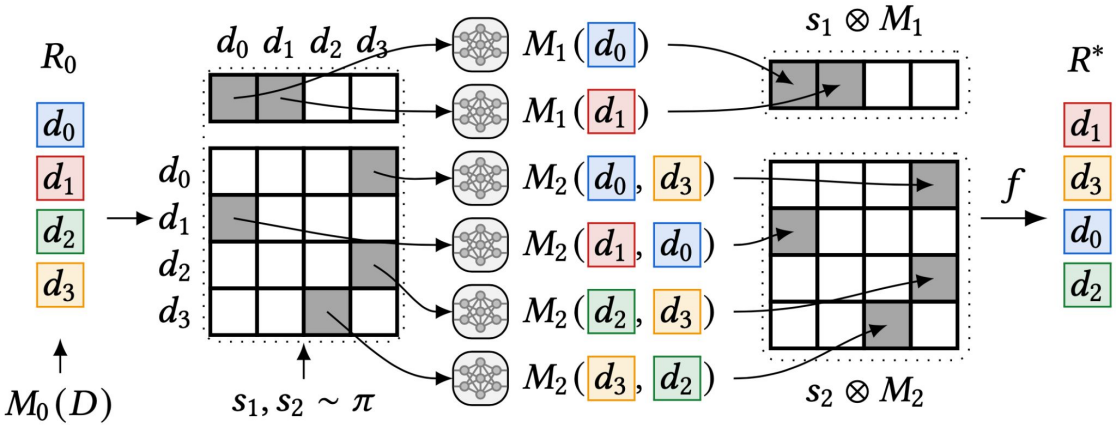
- All parameters are based on the document(s) rank(s) in the first-stage ranking.

The selection policy is optimized with the straight-through estimator:

$$\frac{\delta s_1(d)}{\delta \pi(s_1(d)=1)} = \pi(s_1(d)=1), \quad \frac{\delta s_2(d, d')}{\delta \pi(s_2(d, d')=1)} = \pi(s_2(d, d')=1)$$

Framework: Optimizing Retrieval System Designs

- Our system is **fully differentiable** through **learning-to-rank** and **straight-through estimation**.
- Our model is a **generalization** of (cascading versions) of **its components**.
- We propose a novel **unsupervised distillation loss** that bounds the loss in DCG between rankers.
- Through stochastic-gradient-descent we learn what predictions to select and how to aggregate them, resulting in an **optimized system design of the compound retrieval system**.



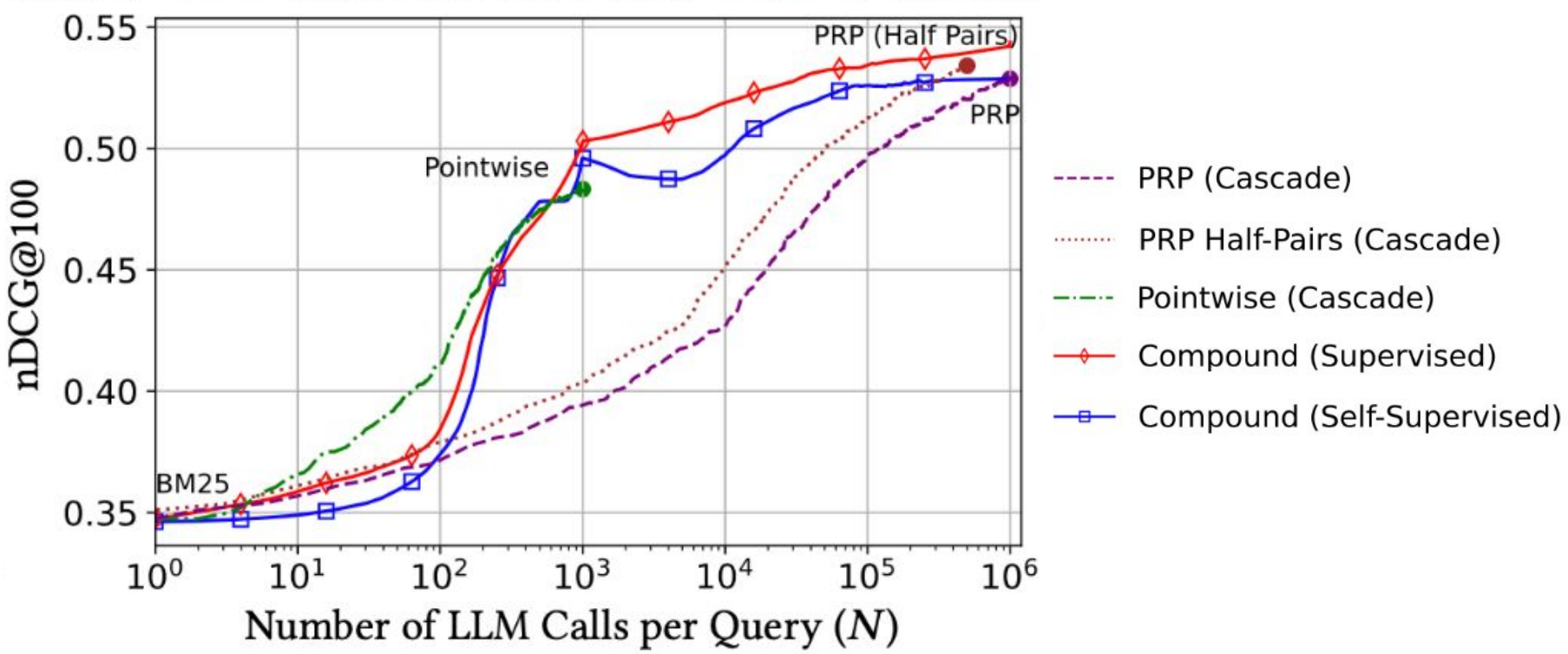
Experimental Setup

- We optimize a system based on the *Gemini 1.5 Flash* LLM on TREC-DL (2019-2022).
- BM25 as first-stage top-1000 ranker, pointwise and pairwise prompts and transformations:

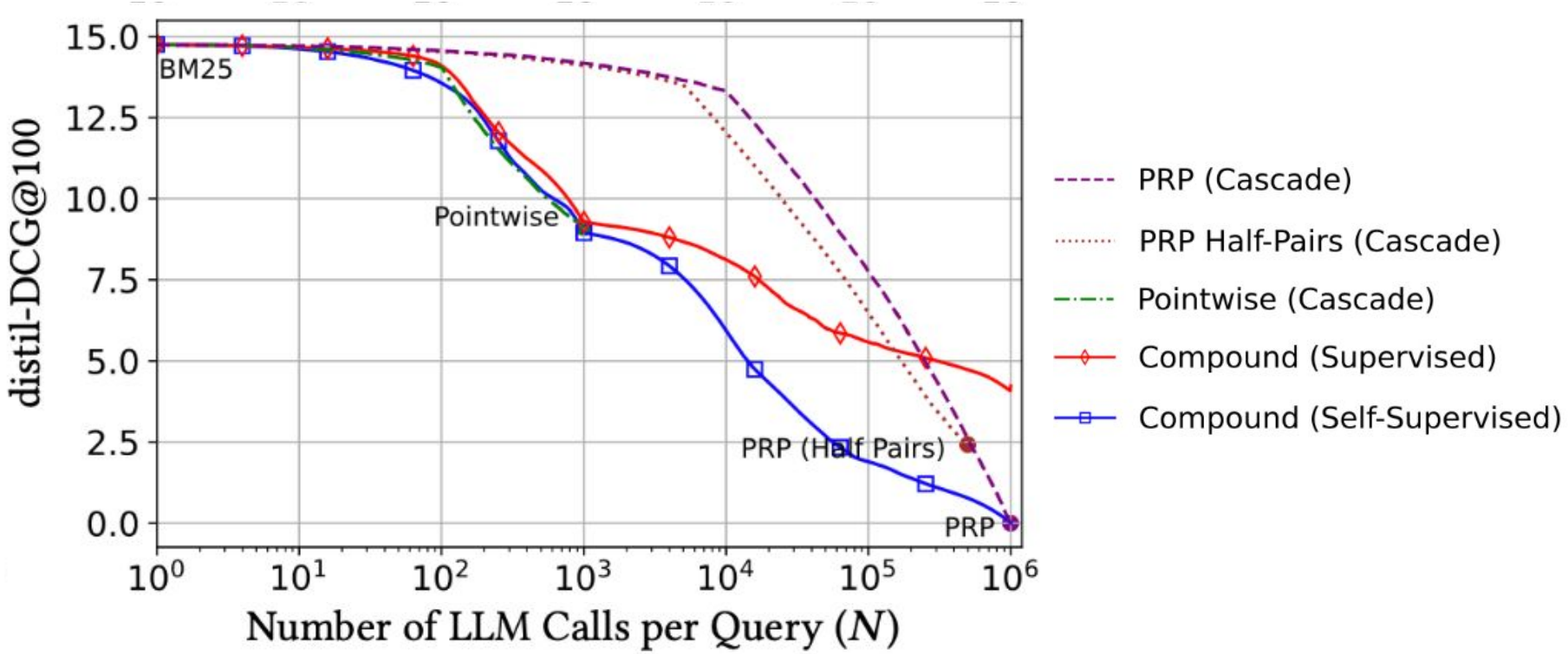
$$
\begin{aligned}
M_0(d) &= \mathrm{BM25}(d), \\
M_1(d) &= \mathrm{LLM}_{\mathrm{point}}(d), & M_2(d) &= \mathrm{round}(M_1(d)), \\
M_3(d, d') &= \mathrm{LLM}_{\mathrm{pair}}(d, d'), & M_4(d, d') &= \mathrm{round}(M_3(d, d')), \\
M_5(d, d') &= 1 - \mathrm{LLM}_{\mathrm{pair}}(d', d), & M_6(d, d') &= \mathrm{round}(M_5(d', d)), \\
M_7(d, d') &= \mathrm{sign}(\mathrm{LLM}_{\mathrm{point}}(d) - \mathrm{LLM}_{\mathrm{point}}(d')).
\end{aligned}
$$

- LTR losses optimize DCG (supervised) and distil-DCG (unsupervised) with @K in {25,100,1000}.
- Policies are made deterministic through a final sampling and maximization step.
- 200 trade-off values are optimized to create trade-off curves,
  50 curves are created on 50 random data-splits and averaged.

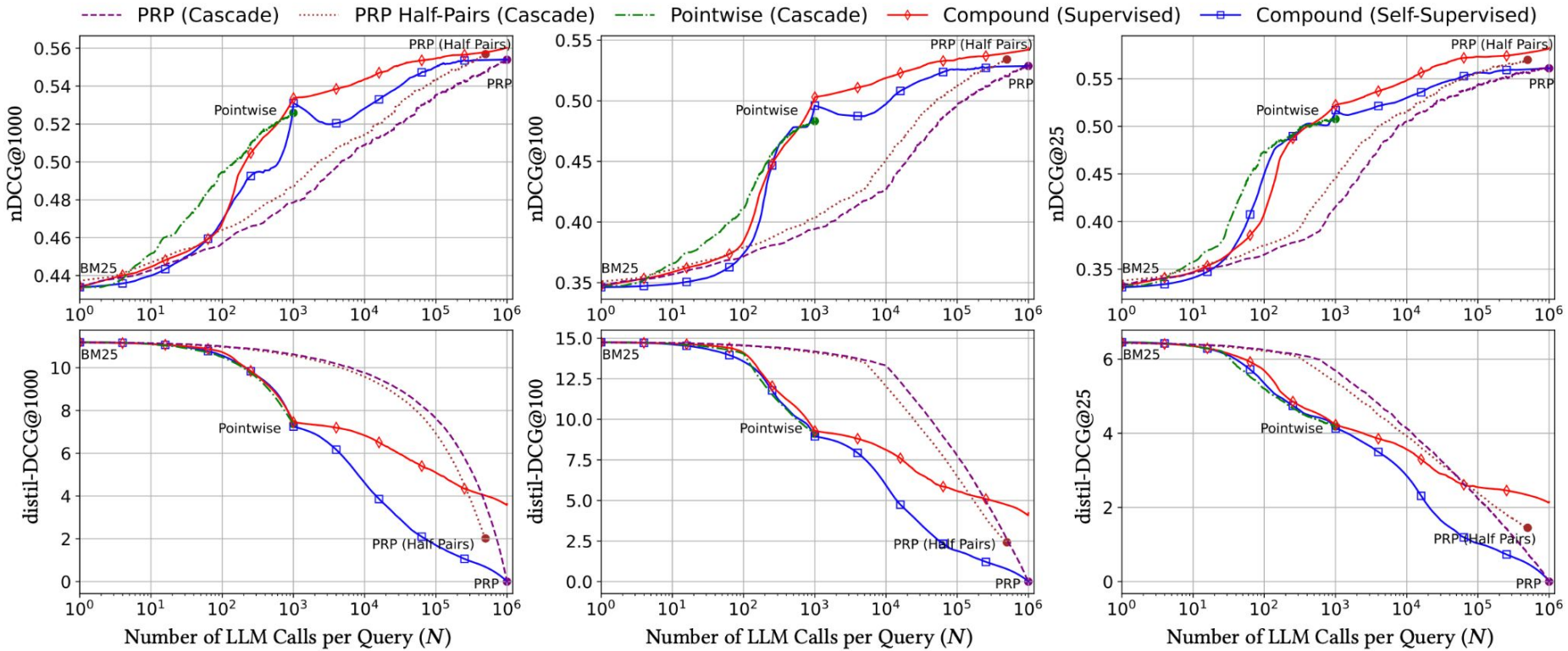
Results: Efficiency-Effectiveness Tradeoffs



Results: Efficiency-Effectiveness Tradeoffs

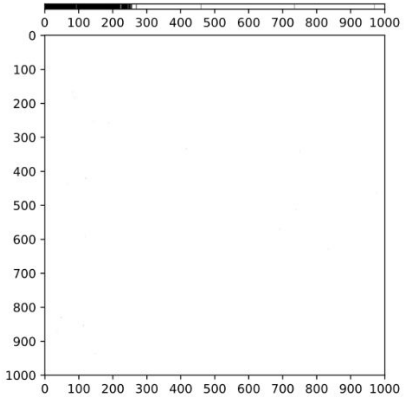


Results: Efficiency-Effectiveness Tradeoffs

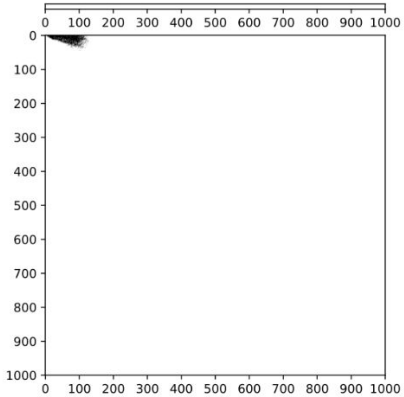


Results: Novel Retrieval System Designs

(a) Distil-DCG@25 loss, $N = 274$.



(b) Distil-DCG@100 loss, $N = 3013$.



Conclusion: Optimizing Compound Retrieval Systems

Contributions

- The **concept** of **compound retrieval systems**, a generalization of cascading retrieval systems.
- A **framework** for the **optimization** of compound retrieval systems **design**:
  - *What predictions to gather from component models?*
  - *How to aggregate the predictions to construct rankings?*
- **Experimental results** that reveal:
  - Better efficiency-effectiveness trade-offs.
  - Completely novel system designs.
- A **novel research direction** in information retrieval system design.
- **LTR can optimize everywhere and all at once!**

Part 3:

Part 3: A Perspective on the Future of Search and Recommender System Optimization



Part 3: A Perspective on the Future of Search and Recommender System Optimization

Conclusion

The Future of Search and RecSys Optimization

- LTR can optimize much **more than relevance objectives**.
- **Reinforcement LTR** can effectively optimize **any exposure-based** objective:
  - Reframing ranking as the **distribution of exposure** is more effective, and makes **implementing** new loss functions much **easier**.
- LTR can **optimize system designs**:
  - Novel designs are **unexplored** and findable with optimization.
- The full potential of LTR is unexploited, **open your mind to the possibilities**.

Thank You for Listening! Questions?



Exposure-Based Reinforcement Learning to Rank

Harrie Oosterhuis*
University of Amsterdam
Amsterdam, NL
h.oosterhuis@uva.nl

Rolf Jagerman
Google DeepMind
New York, US
jagerman@google.com

Zhen Qin
Google DeepMind
New York, US
zhenqin@google.com

Xuanhui Wang
Google DeepMind
Mountain View, US
xuanhui@google.com

Abstract

Reinforcement learning (RL) methods for learning-to-rank (LTR) can optimize (almost) any ranking goal, e.g., from *precision* or *discounted cumulative gain* to *fairness-of-exposure* or *ranking distillation*. However, standard RL is ineffective and computationally costly due to the enormous action space in LTR settings. Existing methods reach computational efficiency through *custom gradient* computation algorithms, but they are very complex to implement and often clash with auto-differentiation. Consequently, existing RL for LTR is not attractive to many practitioners.

We reconsider RL for LTR while actively avoiding reliance on custom gradients. Contrary to the existing approaches, we focus on variance reduction and GPU computation. In doing so, we discover that high sample-efficiency can be reached through *baseline corrections* and *partial marginalization*. Furthermore, we propose an abstraction that places gradient estimation behind a document-exposure distribution, this enables seamless *plug-and-play* integration with auto-differentiation. Thereby, one only has to implement a loss as a differentiable function of exposure and RL for LTR can optimize it using auto-differentiation.

Our experimental results reveal that our new exposure-based RL for LTR approach converges considerably faster and at significantly higher ranking performance than existing custom gradients, with no additional costs in computation time when using GPUs. In contrast, existing custom gradients result in severe stability issues when converging over many epochs, which never occur for our methods. Thus, we considerably improve RL for LTR methodology by increasing its effectiveness, efficiency, and ease of application.

CCS Concepts

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

Keywords

Learning to Rank, Reinforcement Learning, Auto-Differentiation

ACM Reference Format:

Harrie Oosterhuis, Rolf Jagerman, Zhen Qin, and Xuanhui Wang. 2026. Exposure-Based Reinforcement Learning to Rank. In *Proceedings of the 2026 International ACM SIGIR Conference on Innovative Concepts and Theories in Information Retrieval (ICTIR) (ICTIR '26), July 25, 2026, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3805713.3820396

*Work done while Harrie Oosterhuis was working at Google DeepMind.

This work is licensed under a Creative Commons Attribution 4.0 International License.
ICTIR '26, Melbourne, VIC, Australia
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2600-2/2026/07
https://doi.org/10.1145/3805713.3820396

1 Introduction

The optimization of ranking systems is a core problem in the fields of information retrieval (IR) [33] and recommender systems (RecSys) [28]. Its main challenge is that sorting-based metrics result in flat and non-continuous loss functions that provide no useful gradients. Correspondingly, the sub-field of learning-to-rank (LTR) is concerned with surrogate loss functions that can provide usable substitute gradients [24], e.g., through bounding functions [2, 59], approximations [19, 46, 48, 62] or ad-hoc heuristics [9, 43].

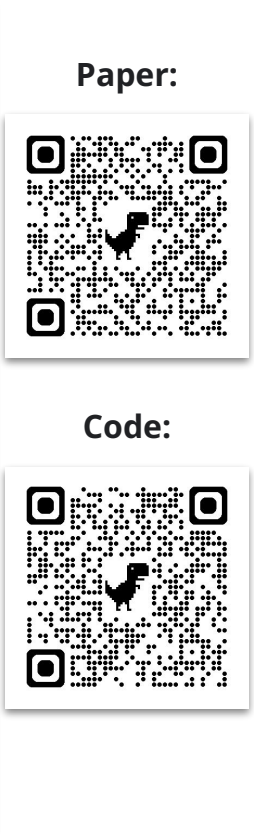
Whilst most LTR methods are specifically designed for traditional relevance metrics (i.e., discounted cumulative gain (DCG) or precision@K) [24, 25, 59], a branch of LTR methods based on reinforcement learning (RL) [54] are uniquely applicable to a much larger class of exposure-based ranking metrics [37, 38, 53]. Here exposure is analogous to the expected attention that a document receives from users due to a ranking model [16]. The origin of this frame is in fairness-of-exposure metrics that aim to fairly distribute attention over sets of documents [4, 16, 52, 53]. However, traditional relevance metrics and certain ranking-distillation losses can also be posed as exposure-based. Thereby, RL for LTR is considerably more versatile than most LTR methodologies [37].

Despite its versatility, RL for LTR is not straightforward to apply; RL struggles with large action spaces which conflicts with the enormous number of possible permutations that can be chosen as rankings in the LTR setting [7, 37, 54]. Even with the extensive use of sampling, standard RL results in high variance and can have high computational costs. Previous work has alleviated this issue by introducing *PL-Rank* – a computationally-efficient algorithm to estimate the policy gradient [37, 38]. In practice, PL-Rank requires the implementation of a complicated custom gradient function, which makes it prone to implementation errors. Additionally, PL-Rank was made to minimize computation time on CPUs, and in doing so, appears to sacrifice numerical stability, resulting in serious convergency issues. Manual implementation of custom gradients also does not integrate well into auto-differentiation frameworks, which have become the standard for machine learning [1, 6, 24, 44].

In this work, we reconsider RL for LTR while focusing on GPU computation and avoiding the usage of custom gradients. Instead, we consider RL techniques such as *baseline corrections* [31, 54] and *partial marginalization* to increase sample-efficiency. Furthermore, we propose an estimator on the exposure distribution [16, 53] instead of the final metric/loss; in implementation, this acts as an abstraction that places the RL policy gradient estimation behind the exposure distribution. Thereby, it seamlessly integrates with auto-differentiation in a *plug-and-play* manner for any exposure-based loss function, as practitioners simply only have to implement a loss as a differentiable function over the exposure distribution. Thus, we avoid the complicated implementation of custom gradients, by abstracting complexity away behind exposure distributions.

Paper:

Code:



Optimizing Compound Retrieval Systems

Harrie Oosterhuis*
Radboud University
Nijmegen, NL
harrie.oosterhuis@ru.nl

Rolf Jagerman
Google DeepMind
New York, US
jagerman@google.com

Zhen Qin
Google DeepMind
New York, US
zhenqin@google.com

Xuanhui Wang
Google DeepMind
Mountain View, US
xuanhui@google.com

Abstract

Modern retrieval systems do not rely on a single ranking model to construct their rankings. Instead, they generally take a cascading approach where a sequence of ranking models are applied in multiple re-ranking stages. Thereby, they balance the quality of the top-K ranking with computational costs by limiting the number of documents each model re-ranks. However, the cascading approach is not the only way models can interact to form a retrieval system.

We propose the concept of *compound retrieval systems* as a broader class of retrieval systems that apply multiple prediction models. This encapsulates cascading models but also allows other types of interactions than top-K re-ranking. In particular, we enable interactions with large language models (LLMs) which can provide relative relevance comparisons. We focus on the optimization of compound retrieval system design which uniquely involves learning where to apply the component models and how to aggregate their predictions into a final ranking. This work shows how our compound approach can combine the classic BM25 retrieval model with state-of-the-art (pairwise) LLM relevance predictions, while optimizing a given ranking metric and efficiency target. Our experimental results show optimized compound retrieval systems provide better trade-offs between effectiveness and efficiency than cascading approaches, even when applied in a self-supervised manner.

With the introduction of compound retrieval systems, we hope to inspire the information retrieval field to more out-of-the-box thinking on how prediction models can interact to form rankings.

CCS Concepts

• **Information systems** → **Search engine architectures and scalability**; **Retrieval models and ranking**; **Learning to rank**.

Keywords

Learning to Rank, Large Language Models, Ranking Distillation

ACM Reference Format:

Harrie Oosterhuis, Rolf Jagerman, Zhen Qin, and Xuanhui Wang. 2025. Optimizing Compound Retrieval Systems. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '25), July 13–18, 2025, Padua, Italy.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3726302.3730051

*Work done while Harrie Oosterhuis was working at Google DeepMind.

This work is licensed under a Creative Commons Attribution International 4.0 License.
SIGIR '25, Padua, Italy
© 2025 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-1592-1/2025/07
https://doi.org/10.1145/3726302.3730051

1 Introduction

Modern retrieval systems consist of many components that are applied together to construct rankings of documents [8]. Thereby, they can combine different prediction models that provide different ranking utility at different costs, such that their service is both responsive and provides high quality search results [18, 38]. Accordingly, it appears very important for the information retrieval (IR) field to consider the following question:

How should retrieval systems utilize multiple prediction models to form rankings in the most effective and efficient manner?

Over the past decade, the IR field appears to have reached a consensus that cascading retrieval systems are the answer to this question [1, 2, 5, 6, 9, 11, 12, 18, 19, 29, 38, 45, 47]. In the cascading paradigm, models are applied in sequence to re-rank the top results of the previous model [38]. The order of the sequence follows model scalability, which generally is inversely correlated with ranking effectiveness [6]. In other words, a retrieval model M_0 with minimal costs per document is used to create a first ranking (e.g., BM25 [33] or dense retrieval [12, 44]), its top-K_0 results are subsequently re-ranked by the next model M_1, its top-K_1 results are then re-ranked by M_2, and so forth. The cascading paradigm applies costlier models to fewer documents than cheaper models, this saves costs whilst it also uses the costlier models to improve upon the rankings of cheaper models [9, 27]. As a result, a cascading retrieval model can have an efficiency-utility trade-off that none of its models can obtain individually [2, 6, 38, 45].

However, the recent introduction of large language models (LLMs) has revealed limitations of the cascade paradigm. LLMs are extremely flexible in their application, as they can handle a virtually endless variety of prompts [4, 36, 39, 46]. Previous work found that using LLMs for predicting relative relevance differences instead of absolute judgements results in substantially better rankings [20, 21, 31, 42]. For instance, *pairwise relevance prompting* (PRP) [31] has state-of-the-art zero-shot ranking by generating pairwise relevance labels that each indicate whether one document is more relevant than another. The downside of standard PRP is that the number of pairs -and thus the number of LLM calls- scales quadratically with the number of documents. This makes PRP extremely costly, even as the final stage in a cascade retrieval system.

Inspired by this shortcoming and because the cascading paradigm has rarely been questioned in the IR field, we reconsider the question stated in the first paragraph. Our first contribution is introducing the concept of a *compound retrieval system* that encompasses any retrieval system that applies multiple predictions models. By keeping our definition broad and avoiding assumptions, we aim to explore the largest design space possible. We argue that compound retrieval system design depends on two essential questions:

(i) *What predictions should the component models make?*
(ii) *How should predictions be combined to construct a ranking?*

Paper:

Code: