Differentiable Permutation Self-Consistency

Arul Saxena
UC Santa Barbara
saxena@ucsb.edu

1 Objectives and Challenges

Large Language Models (LLMs) are increasingly used as *re-rankers* in modern information retrieval (IR) pipelines, but they face key challenges: strong positional biases and the "lost in the middle" effect in long contexts. Permutation Self-Consistency (PSC)[4] was proposed to mitigate these limitations by marginalizing over permutations of the input context. Although the original PSC work included an end-to-end IR pipeline, this code was not publicized.

In this project, we have three core objectives. First, we reproduce the full retrieval–reranking pipeline used in the original PSC study. Second, we examine the challenge of permutation-based aggregation by comparing PSC's Kemeny-optimal procedure with alternative rank aggregation methods. Finally, we address the limitation that PSC operates only at inference time by formulating a differentiable relaxation that can be used during training to reduce positional bias directly through gradient-based learning.

2 State-of-the-Art Techniques Used

Our project builds on several recent advances in LLM reasoning, rank aggregation, and differentiable sorting.

Our overall pipeline—an initial sparse or neural retriever followed by an LLM-based re-ranking stage—builds directly on the framework established by Sun et al in *"Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents"* (EMNLP 2023) .

We also build upon Permutation Self-Consistency (PSC), proposed by Tang et al. in *"Permutation Self-Consistency: Mitigating Positional Bias in Long-Context LLM Reasoning"* (ICLR 2024). PSC marginalizes over random permutations of the input context and has been shown to reduce "lost in the middle" effects during reranking.

For aggregation baselines, we incorporate classical ranking techniques, including Reciprocal Rank Fusion (RRF) by Cormack et al., *"Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods"* (SIGIR 2009), Borda Count, and Tideman's Ranked Pairs method. These methods provide computationally efficient alternatives to Kemeny aggregation.

We leverage NeuralSort, a continuous relaxation of permutation matrices introduced by Grover et al. in *"Stochastic Optimization of Sorting Networks"* (ICLR 2019). NeuralSort provides a differentiable approximation to the sorting operator, enabling gradient-based optimization over ranking structures.

3 Key Algorithms and Techniques Used

3.1 Initial Retrieval

We adopt two standard first-stage retrieval methods. The first is BM25, a sparse lexical matching algorithm widely used in IR pipelines. The second is SPLADE++, a state-of-the-art learned sparse

Preprint. Under review.

retriever that expands queries and documents into high-dimensional sparse vectors using transformer-based encoders.

3.2 LLM-Based Reranking and Aggregation

After first-stage retrieval, the top-k candidate documents are passed to an LLM, which produces a ranking over the candidates. Because individual LLM rankings may be biased or noisy, we evaluate several rank aggregation methods to combine multiple rankings or multiple permutations under PSC.

3.2.1 Kemeny Optimal Aggregation

Kemeny aggregation seeks the ranking $\sigma^\star$ that minimizes the total Kendall-τ distance to all input rankings $\{\pi_1, \ldots, \pi_m\}$. Formally:

$$\sigma^\star = \arg\min_{\sigma} \sum_{i=1}^{m} d_\tau(\sigma, \pi_i).$$

This method is theoretically appealing but NP-hard, making it computationally expensive for large k. However, this problem is fixed-parameter tractable (FPT) with respect to the number of candidates (in this case, k). In other words, Kemeny aggregation is tractable for small values of k.

3.2.2 Borda Count

Borda assigns each document a score equal to the sum of its position-based scores over all rankings. With n documents, document d receives score:

$$\text{Borda}(d) = \sum_{i=1}^{m} (n - \text{rank}_i(d)).$$

3.2.3 Reciprocal Rank Fusion (RRF)

RRF[1] computes a score for each document by summing the reciprocal of its rank across rankings. For document d with rank $\text{rank}_i(d)$ in ranking i:

$$\text{RRF}(d) = \sum_{i=1}^{m} \frac{1}{k + \text{rank}_i(d)}.$$

3.2.4 Tideman's Ranked Pairs

Tideman's Ranked Pairs method constructs a directed graph of pairwise preferences, locking in edges in decreasing order of victory strength while avoiding cycles. Let $P(a, b)$ denote the number of rankings preferring a to b. The pairwise margin is:

$$M(a, b) = P(a, b) - P(b, a).$$

Edges are sorted by $M(a, b)$ and added greedily unless they create a cycle; the topological ordering of the resulting DAG yields the final ranking.

3.3 Differentiable Permutation Self-Consistency (Diff-PSC)

Building on NeuralSort[2], Diff-PSC provides a fully differentiable relaxation of the PSC pipeline by replacing all discrete ranking and aggregation steps with continuous counterparts. The key idea is to compute soft permutation matrices for multiple random input permutations, aggregate their induced pairwise ordering probabilities, and use these to construct a differentiable self-consistent ranking objective.

Notation. Let $X = \{X_1, \ldots, X_n\}$ be the items to rank, $\sigma^\star$ the ground-truth ranking, and $h_\theta : \mathcal{X}^n \to \mathbb{R}^n$ a parametric scorer. Let $\Pi = \{\pi_1, \ldots, \pi_m\}$ be m i.i.d. permutations sampled uniformly from S_n. Fix a NeuralSort temperature $\tau > 0$ and define the rank index vector $r = (1, 2, \ldots, n)^\top \in \mathbb{R}^n$.

Diff-PSC Procedure. For each permutation $\pi_i \in \Pi$:

$$\text{Compute model scores:} \qquad s^{(i)} = h_\theta\big(X[\pi_i]\big) \in \mathbb{R}^n, \quad i = 1, \dots, m. \tag{1}$$

$$\text{Compute pairwise differences:} \qquad A^{(i)}_{p,q} = |s^{(i)}_p - s^{(i)}_q|, \quad p, q = 1, \dots, n. \tag{2}$$

$$\text{Row sum for centering:} \qquad b^{(i)}_p = \sum_{q=1}^{n} A^{(i)}_{p,q}, \quad p = 1, \dots, n. \tag{3}$$

$$\text{Position weights:} \qquad v_k = n + 1 - 2k, \quad k = 1, \dots, n. \tag{4}$$

$$\text{NeuralSort score matrix:} \qquad M^{(i)}_{p,k} = s^{(i)}_p \, v_k - b^{(i)}_p, \quad p, k = 1, \dots, n. \tag{5}$$

$$\text{Soft permutation (row-wise softmax):} \qquad P^{(i)}_{p,k} = \frac{\exp(M^{(i)}_{p,k}/\tau)}{\sum_{t=1}^{n} \exp(M^{(i)}_{p,t}/\tau)}. \tag{6}$$

Soft pairwise ordering probabilities. For each item pair $a \neq b$:

$$p^{(i)}_{ab} = \sum_{r=1}^{n} \sum_{s=1}^{r-1} P^{(i)}_{a,r} P^{(i)}_{b,s}, \tag{7}$$

representing the probability that item a is ranked before b under permutation π_i.

Aggregate across permutations:

$$\bar{p}_{ab} = \frac{1}{m} \sum_{i=1}^{m} p^{(i)}_{ab}. \tag{8}$$

Compute aggregated item scores:

$$\tilde{s}_j = \sum_{k \neq j} \bar{p}_{jk}, \quad j = 1, \dots, n. \tag{9}$$

Optional final soft central permutation via NeuralSort:

$$A^\star_{p,q} = |\tilde{s}_p - \tilde{s}_q|, \qquad b^\star_p = \sum_q A^\star_{p,q}, \tag{10}$$

$$M^\star_{p,k} = \tilde{s}_p \, v_k - b^\star_p, \qquad P^\star_{p,k} = \frac{\exp(M^\star_{p,k}/\tau)}{\sum_{t=1}^{n} \exp(M^\star_{p,t}/\tau)}. \tag{11}$$

Training objective (pairwise cross-entropy). Given ground-truth ranking $\sigma^\star$:

$$\mathcal{L}_{\text{DiffPSC}}(\theta) = -\sum_{a<b} \Big[\mathbf{1}\{\sigma^\star(a) < \sigma^\star(b)\} \log \bar{p}_{ab} + \mathbf{1}\{\sigma^\star(b) < \sigma^\star(a)\} \log(1 - \bar{p}_{ab}) \Big]. \tag{12}$$

Backpropagation flows naturally through $\bar{p}_{ab} \leftarrow p^{(i)}_{ab} \leftarrow P^{(i)} \leftarrow s^{(i)} \leftarrow \theta$, enabling end-to-end optimization of a PSC-like objective.

4 Data set + metrics + evaluation

We follow the original PSC paper and adopt the same datasets, evaluation protocol, and metrics to ensure comparability. Experiments are conducted on the MS MARCO passage ranking benchmark and evaluated on TREC Deep Learning 2019 and TREC Deep Learning 2020 test sets. These datasets

Table 1: nDCG@10 results on TREC-DL19 and TREC-DL20

TREC-DL19

First Stage	**Top-k**	**Method**	Original	Kemeny	Borda	Tideman	RRF	Diff-PSC
None	All	BM25	50.58	–	–	–	–	–
	All	SPLADE++	73.08	–	–	–	-	–
BM25	20	GPT-3.5	60.95	61.49	60.81	61.13	61.42	61.36
	20	GPT-4	60.88	64.91	62.47	63.83	64.52	64.19
	20	GPT-5	65.42	**68.75**	66.12	66.41	68.53	68.18
SPLADE++	20	GPT-3.5	69.62	71.53	70.04	69.96	71.18	70.82
	20	GPT-4	73.21	76.85	74.12	74.71	76.28	74.99
	20	GPT-5	76.87	**82.32**	78.48	78.05	79.79	79.01

TREC-DL20

First Stage	**Top-k**	**Method**	Original	Kemeny	Borda	Tideman	RRF	Diff-PSC
None	All	BM25	47.96	–	–	–	–	–
	All	SPLADE++	71.97	–	–	–	–	–
BM25	20	GPT-3.5	57.64	59.61	58.03	58.47	59.48	59.22
	20	GPT-4	57.78	62.48	60.19	60.98	62.12	61.81
	20	GPT-5	62.31	**65.81**	63.87	64.49	65.57	65.28
SPLADE++	20	GPT-3.5	68.51	70.19	69.11	69.48	70.08	69.79
	20	GPT-4	71.97	78.53	73.48	74.98	77.18	76.81
	20	GPT-5	75.52	**81.21**	76.83	77.99	79.81	79.09

contain manually annotated relevance judgments and are standard for benchmarking passage retrieval methods.

We report normalized discounted cumulative gains at rank 10 (nDCG@10) as our primary metric. nDCG@10 is widely used in retrieval tasks and is particularly well-suited for evaluating re-ranking systems where high-quality top-ranked documents are most important.

Across both DL19 and DL20, we observe consistent improvements when applying PSC over the one-shot LLM rankings, with the strongest gains appearing in higher-capacity models. GPT-5 variants show the overall highest nDCG@10.

Among aggregations methods, Kemeny consistently performs best, while RRF and Borda remain promising lightweight alternatives: despite their far lower computational cost, they maintain competitive (though still weaker) performance. SPLADE++ first-stage retrieval further improves all downstream methods. Finally, our Diff-PSC results suggest that it is comparable in performance to PSC while being fully differentiable. Diff-PSC could therefore be incorporated during training-time rather than solely at inference time, offering a compelling direction for future work.

5 Our Contributions

Our core technical contributions are enumerated as follows:

1. **Retrieval Layer.** Implemented BM25 and SPLADE retrieval using Pyserini and integrated MS MARCO + TREC DL19/20 into a unified pipeline.
2. **End-to-End PSC Pipeline.** Reconstructed the full PSC inference pipeline (retrieval $\rightarrow$ LLM scoring $\rightarrow$ permutation sampling $\rightarrow$ aggregation) based on the partial code released by Tang et al.
3. **Experiment Framework.** Built a modular framework supporting multiple GPT models and interchangeable aggregation modules for consistent evaluation.
4. **Additional Aggregation Baselines.** Added RRF, Borda, and Tideman as alternative rank aggregation methods for comparison against PSC.
5. **Diff-PSC.** Introduced a differentiable PSC formulation using NeuralSort, enabling PSC-style consistency to be used during training.

In addition to the technical components above, our project required significant effort in understanding and interpreting the PSC paper's method. Because the official code release was partial and omitted several key steps, we closely studied both the paper and the available implementation to reconstruct the full pipeline.

Finally, this projecty directly connects to th LLM Ranking portion of the course, where we examined how large language models can participate in different stages of the IR pipeline. PSC and our extensions upon it fit within this area, and offer a deeper exploration of the LLM-based ranking methods discussed in class.

References

[1] Gordon V Cormack, Charles LA Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 758–759, 2009.

[2] Aditya Grover, Eric Wang, Aaron Zweig, and Stefano Ermon. Stochastic optimization of sorting networks via continuous relaxations, 2019.

[3] Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. Is chatgpt good at search? investigating large language models as re-ranking agents, 2024.

[4] Raphael Tang, Xinyu Zhang, Xueguang Ma, Jimmy Lin, and Ferhan Ture. Found in the middle: Permutation self-consistency improves listwise ranking in large language models, 2024.