Found in the Middle: Permutation Self-Consistency Improves Listwise Ranking in Large Language Models

Raphael Tang,[*1] **Xinyu Zhang,**[*2] **Xueguang Ma,**[2] **Jimmy Lin,**[2] **Ferhan Ture**[1]
[1]Comcast AI Technologies [2]University of Waterloo
[1]{raphael_tang, ferhan_ture}@comcast.com [2]{x978zhan, x93ma, jimmylin}@uwaterloo.ca

Abstract

Large language models (LLMs) exhibit positional bias in how they use context, which especially affects listwise ranking. To address this, we propose *permutation self-consistency*, a form of self-consistency over the ranking list outputs of black-box LLMs. Our key idea is to marginalize out different list orders in the prompt to produce an order-independent ranking with less positional bias. First, given some input prompt, we repeatedly shuffle the list in the prompt and pass it through the LLM while holding the instructions the same. Next, we aggregate the resulting sample of rankings by computing the central ranking closest in distance to all of them, marginalizing out prompt order biases in the process. Theoretically, we prove the robustness of our method, showing convergence to the true ranking under random perturbations. Empirically, on five datasets in sorting and passage reranking, our approach improves scores from conventional inference by up to 34–52% for Mistral, 7–18% for GPT-3.5, 8–16% for LLaMA v2 (70B). Our code is at https://github.com/castorini/perm-sc.

1 Introduction

Large language models (LLMs) respond cogently to free-form textual prompts and represent the state of the art across many tasks (Zhao et al., 2023). Their quality, however, varies with nuisance positional factors such as prompt order and input length. As a descriptive example, consider this prompt:

> *Arrange the following passages in decreasing relevance to the query, "what are shrews?"*
> *(1) Cats hunt small mammals, such as shrews ...*
> *(2) Shrews are mole-like mammals, widely ...*
> *(3) Shrews use their noses to find prey and ...*

The correct output order is $(2, 3, 1)$, from most to least relevant, but several positional biases may interfere with the model. Liu et al. (2023) demonstrate that LLMs tend to get "lost in the middle" of a long context and use the middle portion poorly, which suggests that the middle passage (2) in the example may get misranked (e.g., $3, 1, \underline{2}$). Wang et al. (2023a) find prompt order to affect quality, with some orders outperforming others; if items 1 and 3 were swapped in the prompt, the LLM would perhaps generate the mistaken ranking $(2, \underline{1}, \underline{3})$.

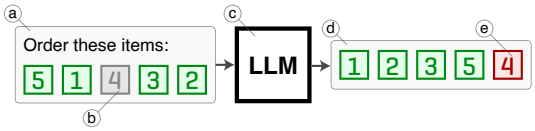


Figure 1: The conventional decoding process for listwise ranking with input prompt (a), language model (c), and output ranking (d). The grey item (b) is "lost in the middle" by the LLM, resulting in its misranking (e).

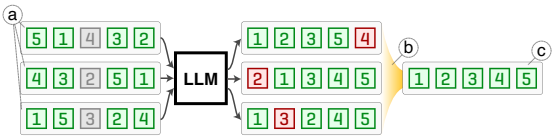


Figure 2: Our permutation self-consistency process. With the instruction fixed, we shuffle the input list for prompts (a), producing outputs with different mistakes. We aggregate (b) these output rankings into one (c).

In this paper, we mitigate positional biases for listwise-ranking LLMs. We propose *permutation self-consistency*, a novel decoding strategy for improving the quality, consistency, and prompt-order invariance of black-box LLMs. First, we construct prompts with randomly permuted input lists, then feed them into an LLM to generate a set of output rankings. Then, we aggregate these outputs into the central ranking that minimizes the Kendall tau distance to all of them, marginalizing out prompt order as a factor; see Figures 1 and 2. As related work, Stoehr et al. (2023) train direction-unaware probes on the representations of language models to detect order consistency, but their evaluation reveals the ranking direction of test examples to the model, deviating from standard practices.

[*] Equal contribution.

Next, we assess the effectiveness of permutation self-consistency, both theoretically and empirically. Theoretically, we prove in Section 2.3 that it recovers the true ranking under arbitrary noise distributions with enough observations and at least one correctly ordered pair in each observation. Experimentally, we apply our method to tasks in math and word sorting, sentence ordering, and passage reranking (Craswell et al., 2020, 2021), consistently increasing the scores of GPT-3.5, GPT-4, and LLaMA v2 (70B; Touvron et al., 2023) by up to 4–17%, 9–24%, and 8–16%, respectively. We achieve similar gains for Mistral (Jiang et al., 2023) and Zephyr (Tunstall et al., 2023). We conclude that permutation self-consistency improves listwise ranking in LLMs. In line with our premises, we observe positional bias, as shown in Section 3.2.

Finally, we conduct auxiliary analyses to justify our design choices. In Section 4.1, our hyperparameter study finds that quality quickly rises with the number of aggregated output rankings: the score improvement from using five aggregated rankings reaches 67% of twenty, on average, suggesting that a few suffice for quality gain. We further demonstrate that sampling temperature is ineffective for us, unlike the original self-consistency work (Wang et al., 2023b) in chain-of-thought reasoning, likely because listwise ranking does not require exploration of various reasoning paths.

Our contributions are as follows: **(1)** we propose a novel decoding technique for improving the quality, consistency, and position invariance of black-box, listwise-ranking LLMs; **(2)** we empirically establish the validity of our method in sorting and passage reranking on seven models and five datasets, and we theoretically prove the robustness of our method to certain classes of ranking noise, including "lost-in-the-middle" type ones; and **(3)** we provide new analyses on positional biases in listwise-ranking LLMs, finding that biases depend on pairwise positions of items in the list.

2 Our Approach

2.1 Preliminaries

Notation. We define an n-*ranking* as a permutation $\sigma : \{1, \ldots, n\} \mapsto \{1, \ldots, n\}$. For some sequence $\boldsymbol{X} := \{X_i\}_{i=1}^n$, define $\boldsymbol{X}[\sigma]$ as the permuted sequence of $\boldsymbol{X}$ transformed by σ, where $\boldsymbol{X}[\sigma]_i := X_{\sigma(i)}$. Let the inversion vector of σ be

$$\mathrm{inv}(\sigma)_i := \#\{j : \sigma(j) > \sigma(i), j < i\}. \quad (1)$$

To quantify dissimilarity, the Kendall tau distance between two rankings σ_1 and σ_2 is the number of inversions in $\sigma_1^{-1} \circ \sigma_2$:

$$d_\kappa(\sigma_1, \sigma_2) := \sum_{i=1}^{n} \mathrm{inv}(\sigma_1^{-1} \circ \sigma_2)_i. \quad (2)$$

In other words, it is the number of pairwise disagreements, or *discordant* pairs, in the permutation ordering. The distance is one affine transform away from the Kendall tau correlation, used to measure list order similarity (Kendall, 1948):

$$\tau(\sigma_1, \sigma_2) := 1 - \frac{2d_\kappa(\sigma_1, \sigma_2)}{\binom{n}{2}}. \quad (3)$$

In the extreme, $\tau = 1 \iff \sigma_1 = \sigma_2$, and $\tau = -1$ implies that one is the other's reverse.

2.2 Permutation Self-Consistency

How do we mitigate positional biases in listwise-ranking LLMs? We find inspiration in the self-consistency framework (Wang et al., 2023b), which improves quality and consistency in chain-of-thought prompting (Wei et al., 2022). The approach has two main stages: first, it *samples* multiple answers for an input prompt; then, it *aggregates* the sampled answers into a single, high-quality one, hence "marginalizing out" separate reasoning paths from the language model.

Unfortunately, self-consistency does not readily generalize to listwise ranking for a few reasons. For one, it is limited to point predictions, greatly simplifying the aggregation procedure to taking the majority vote. For another, sampling temperature, the method's mainstay of generating diverse samples for aggregation, has little effect on (and at times harming) the quality of aggregated predictions in listwise ranking, as shown in Section 4.1. Lastly, self-consistency does not explicitly address positional bias, the central issue of our paper.

Nevertheless, its shuffle–aggregate paradigm is still a useful template. With it, we propose *permutation self-consistency*: for the first sample step, we randomly shuffle the list in the prompt to curate a diverse set of rankings, each with different position biases. For the next aggregate step, we compute the central ranking closest in Kendall tau distance to all the sampled rankings, which, like self-consistency, marginalizes out the independent variable (in the original, reasoning paths; in ours, prompt order). Intuitively, we intervene on list order, collect output rankings, then aggregate, breaking the association between individual list order and output rankings.

Task	Example Input Prompt
Math Sorting	Sort these expressions: 3 / 2, 1 - 5, ...
Sentence Ordering	Order the shuffled sentences: [1] The...
Passage Ranking	Order these by relevance to the query, "what are shrews?": [1] Cats hunt...

Table 1: Listwise-ranking input prompt examples.

Formally, we are given an input sequence of items $\boldsymbol{X} := \{X_i\}_{i=1}^n$, such as a list of passages, along with a listwise-ranking LLM $h(\boldsymbol{X};\ s)$ that returns an n-ranking on some string prompt s; see Table 1 for an example. First, we construct a diverse set of output rankings by randomly permuting $\boldsymbol{X}$ and passing it through the LLM, like how self-consistency uses temperature to vary their output. Specifically, we sample a sequence

$$\hat{\sigma}_i := h(\boldsymbol{X}[\pi_i];\ s) \text{ for } 1 \leq i \leq m, \tag{4}$$

where π_i is drawn uniformly at random from the set of all possible n-rankings. As noted previously, each output ranking has positional bias, but mistakes are expected to differ among the outputs because of our input order randomization. We then "marginalize out" these individual biases by aggregating the output rankings into a single central ranking. One method with attractive theoretical properties is the Kemeny–Young (Kemeny, 1959) optimal ranking of the outputs—that is, the central ranking that minimizes the sum of its Kendall tau distances to every output ranking:

$$\bar{\sigma} := \operatorname*{argmin}_{\sigma} \sum_{1 \leq i \leq m} d_\kappa(\hat{\sigma}_i, \sigma). \tag{5}$$

Our approach returns $\bar{\sigma}$ as the prediction for $\boldsymbol{X}$ and terminates. Although this calculation is NP-hard, fast exact and approximate algorithms exist (Conitzer et al., 2006; Ali and Meilă, 2012), many implemented in our codebase.

Passage reranking. The task of passage ranking is to rank a set of provided passages in order of relevance to a given query. The use of permutation self-consistency for this case deserves special attention. Due to the LLM input length constraint, predominant LLM-based approaches such as RankGPT (Sun et al., 2023), LRL (Ma et al., 2023b), and RankVicuna (Pradeep et al., 2023) stride the LLM across fixed windows of items from the back of the list to the front, rather than output a ranking in a single pass. In this case, we apply permutation self-consistency to each window.

2.3 Theoretical Guarantees

We now show that for certain kinds of noisy rankings, the Kemeny ranking can recover the true ranking given enough observations. For example, if there always exists some random pair of items that is correctly ranked among randomly ordered observations, we will converge to the true ranking.

Definition 2.1. *For two rankings σ_1 and σ_2, the concordant subset is a set S' where $\forall i$ and $j \in S', \sigma_1(i) < \sigma_1(j) \wedge \sigma_2(i) < \sigma_2(j)$ or $\sigma_1(i) > \sigma_1(j) \wedge \sigma_2(i) > \sigma_2(j)$.*

Proposition 2.1. *Let there be a true ranking σ and a sequence of i.i.d. uniformly noisy rankings $\hat{\boldsymbol{\sigma}} := \{\hat{\sigma}_i\}_{i=1}^m$. Suppose each noisy ranking $\hat{\sigma}_k$ has a uniformly random, nonempty concordant subset S'_k with σ, and the remaining rank elements not in S'_k represent a random permutation. Then the Kemeny–Young ranking $\bar{\sigma}$ of $\hat{\boldsymbol{\sigma}}$ converges in probability to σ, i.e., it is a consistent estimator.*

Proof sketch. Let A_{ij} be the event that the sum of discordant pairs indexed by i and j between $\hat{\boldsymbol{\sigma}}$ and σ is greater than the number of concordant ones. $\mathbb{P}(A_{ij})$ is upper-bounded by $\exp(-O(m))$. The union bound of $\mathbb{P}(\bigcap_{i,j} A_{ij})$ shows that the probability of the sum of discordant pairs being greater than that of the concordant pairs vanishes for any pair as m approaches infinity. Thus, the Kemeny-optimal ranking will always approach σ for $m \to \infty$, concluding our proof. □

To extend this, we prove that, in the presence of ranking noise, characterized empirically in Section 3.2, our approach yields a consistent estimator for the true ranking, given that at least one possibly nonrandom pair of items is always concordant:

Proposition 2.2. *Let there be a true ranking σ and a distribution of noisy rankings $\mathbb{P}(\sigma_{noise})$, where $\sigma_{noise} \circ \pi$ always has a uniform, non-empty concordant subset S with σ for any input ranking π, and the elements not in S are uniformly random. Then the permutation self-consistency procedure is a consistent estimator of σ when applied to the input π and the "LLM" characterized by $\mathbb{P}(\sigma_{noise})$.*

Proof sketch. Observe that the first shuffling stage of permutation self-consistency transforms the premises into those of Proposition 2.1. Since the next stage of the method involves the same Kemeny–Young ranking as the proposition does, the rest of the proof quickly follows. □

Full proofs are in Appendix A.

1. MathSort: Sort ten arithmetic expressions by value.
Example: `3 / 5, 2 - 9, 6 * 5, 2 * 1, 3 / 1, 9 * 9, 1 - 9, 9 + 8, 3 / 5, 1 / 9.`
2. WordSort: Order ten words alphabetically.
Example: `aaron, roam, aardvark, nexus, [...].`
3. GSM8KSort: Unscramble sentences from GSM8K.

Example: Order the scrambled sentences logically:
`- She took 1 hour to walk the first 4 miles [...]`
`- Marissa is hiking a 12-mile trail.`
`- If she wants her average speed to be 4 [...]`

Table 2: Example prompts for our three sorting tasks.

Method	MATHSORT		WORDSORT		GSM8KSORT	
	Orig.	PSC	Orig.	PSC	Orig.	PSC
Mistral-7B	34.7	52.9	55.3	74.2	46.7	65.3
Zephyr$_\beta$-7B	13.2	32.2	30.7	60.8	34.5	61.6
LLaMA$_2$-7B	8.7	24.2	41.3	59.9	6.1	21.3
LLaMA$_2$-13B	16.7	26.0	65.4	78.8	42.7	46.8
LLaMA$_2$-70B	27.9	31.3	74.6	81.0	61.1	71.2
GPT-3.5	64.0	75.2	85.9	88.1	82.1	88.4
GPT-4	83.5	**89.6**	89.9	**92.0**	88.4	**90.5**

Table 3: Kendall tau correlation scores on our sorting tasks. Original scores are the median across 20 single runs, and PSC aggregates those 20. Underline indicates improvement from PSC and bold denotes best.

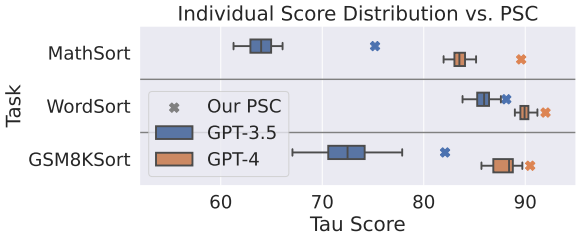


Figure 3: The distribution of sorting task scores from twenty individual runs plotted against our PSC score. Our PSC outperforms the best of any individual run.

3 Experiments

We experiment on sorting and passage ranking, two distinct types of problems in listwise ranking.

3.1 Sorting Tasks

Setup. We build three functionally distinct datasets called MathSort, WordSort, and GSM8KSort, corresponding to numerical sorting, alphabetical ordering, and sentence arrangement, respectively. For MathSort, the task is to sort ten random mathematical expressions of the form `digit op digit`, where `digit` is a single digit and `op` is one of `+`, `-`, `*`, or `/`. In WordSort, the goal is to order ten random English words alphabetically. Finally, GSM8KSort is a sentence-unscrambling task over the test set of the GSM8K reasoning dataset (Cobbe et al., 2021). For consistency and tractability, we use 100 examples in each dataset; see Table 2 for prompts.

These synthetic sorting datasets have certain benefits. The items are intrinsically comparable, especially in MathSort and WordSort, whose elements have unequivocal order (e.g., "aardvark" must precede "abacus" in WordSort). On the other hand, passage ranking relies on human judgment, where label noise may confound findings. Synthetic construction also enables control of item length: MathSort examples are fixed at three tokens, WordSort at a single word, and GSM8K one sentence.

For our LLMs, we choose the open families of LLaMA v2 models (Touvron et al., 2023), Mistral-7B Instruct (Jiang et al., 2023), and Zephyr$_\beta$-7B (Tunstall et al., 2023), along with the closed GPT-3.5 (Turbo, the "0613" version) and GPT-4 from OpenAI, both the state of the art. We apply permutation self-consistency with $m = 20$ output rankings, resulting in 20 parallel calls to the LLM per example. Detailed settings are in Appendix B.2.

Results. We present our main results in Table 3, naming our method "PSC" for short. PSC consistently outperforms conventional inference on all three datasets and seven models by an average of 51% in Kendall tau correlation, skewed toward the smaller variants. Specifically, LLaMA$_2$-7B, 13B, and 70B attain average score increases of 157%, 28%, and 12%, respectively, Mistral and Zephyr improve by 42% and 106%, and GPT-3.5 and GPT-4 by 3–18% and 2–7%. We attribute this to the already high quality of the larger 70B and GPT models, which leave less room for improvement. Task-wise, we improve MathSort, WordSort, and GSM8KSort by 67%, 30%, and 58%, and gains negatively correlate with original quality ($r = -0.72$). We conclude that PSC improves listwise ranking on sorting tasks, with higher gains on smaller models and more difficult tasks.

One foreseeable question is whether any individual runs surpass PSC, which would weaken the case for rank aggregation. To answer this, we plot the distribution of the individual scores against PSC in Figure 3. We observe that PSC reliably beats all individual runs by 1–12%, improving the most on tasks and models with lower baseline quality, such as MathSort and GPT-3.5. These findings bolster the necessity of the aggregation step.

First Stage	Top-k	Method	TREC-DL19		TREC-DL20	
			Original	Our PSC	Original	Our PSC
None	All	(1) BM25	50.58	–	47.96	–
	All	(2) SPLADE++ ED	73.08	–	71.97	–
Supervised Approaches						
BM25	100	(3) MonoT5 (T5-3B)	71.83	–	68.89	–
	100	(4) RankT5 (T5-3B)	71.22	–	69.49	–
	100	(5) RankLLaMA (13B)	73.22	–	70.38	–
Unsupervised Approaches						
BM25	100	(6) PRP-Best (FLAN-T5-XXL)	69.87	–	69.85	–
	100	(7) PRP-Best (FLAN-UL2)	72.65	–	70.68	–
	100	(8) RankVicuna	66.83	68.70	65.49	65.68
	20	(9) Single (GPT-3.5)	60.95 (60.96)	61.49	57.64 (57.68)	59.62
	20	(10) Single (GPT-4)	60.88 (60.92)	64.88	57.78 (57.89)	62.49
	100	(11) RankGPT (GPT-3.5)	68.00 (68.13)	70.77	62.08 (63.20)	62.70
	100	(12) RankGPT (GPT-4)	75.00 (75.59)	**75.66**	70.36 (70.56)	**71.00**
SPLADE++ ED	100	(13) RankVicuna	74.59	74.13	74.73	74.06
	20	(14) Single (GPT-4)	73.21 (73.36)	**76.87**	71.97 (73.63)	**78.52**
	100	(15) RankGPT (GPT-4)	74.64 (74.93)	76.01	70.76 (71.08)	75.14

Table 4: nDCG@10 results on DL19 and 20. The maximum across three runs are in parentheses, while those outside the median. Improvements from PSC are underlined and best per section are bolded. On the one-tailed signed-rank test, paired differences between the original and PSC are significant at the 99% confidence level ($p < 0.01$).

3.2 Passage Reranking Task

For a longer-context task, we evaluate our method on passage reranking. For a query and an initial list of relevant documents from a fast, first-stage retriever, we must reorder the documents so that more relevant ones come first.

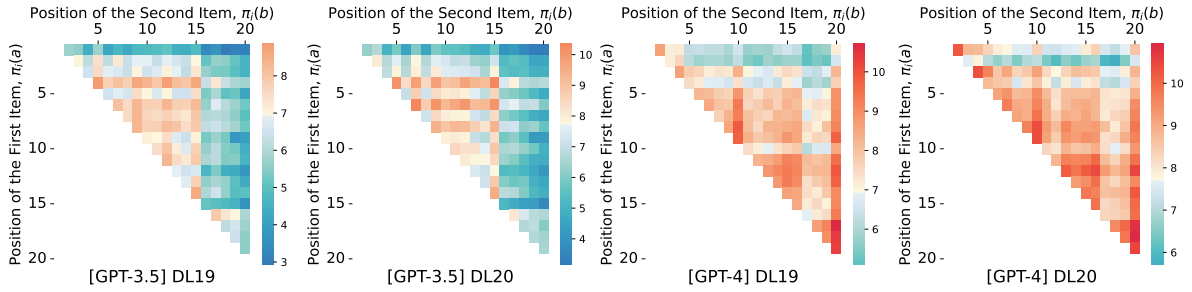
Setup. We select the passage retrieval test sets from the TREC Deep Learning Tracks DL19 and DL20 (Craswell et al., 2020, 2021), both canon in the literature (Qin et al., 2023). These datasets are built on the MS MARCO v1 corpus (Bajaj et al., 2016), which contains 8.8 million passages. As is standard, we rerank the top-100 passages retrieved by the first-stage BM25 (Robertson et al., 2009) or SPLADE++ EnsembleDistill (ED; Formal et al., 2021), reporting nDCG@10 scores for quality.

Like sorting, we pick an open LLM, RankVicuna (Pradeep et al., 2023), fine-tuned from Vicuna (Chiang et al., 2023), and a closed family, GPT-3.5 and GPT-4—all models match state of the art. RankVicuna and GPT-3.5 have context lengths of 4096, half of GPT-4's 8192. We similarly apply permutation self-consistency with $m = 20$ runs. Furthermore, for three of our variants named "single," we reduce the top-100 to 20 and discard the windowing strategy used in RankGPT and RankVicuna, described in Section 2.2. This allows us to fit all passages in a single call and thus remove potentially confounding interactions between the windowing method and permutation self-consistency.

For our supervised baselines, we report results from the MonoT5 (Nogueira et al., 2020) and RankT5 (Zhuang et al., 2023) models, based on the T5 language model (Raffel et al., 2020). We also run RankLLaMA (Ma et al., 2023a), the current pointwise state of the art. For the unsupervised baselines, we copy figures from the state-of-the-art pairwise ranking results across the variants in Qin et al. (2023), which we name PRP-Best for short.

Results. We present our results in Table 4. Our PSC outperforms all conventional inference baselines: first, RankGPT with PSC on DL19 (row 12) edges ahead by 0.07 points (same row); second, the same for DL20 (row 12), leading PRP by 0.32 points (row 7); third, the overall top result on DL19 of 76.87 from SPLADE++ (row 14), outperforming the previous by 1.28 (row 12); and fourth, 78.52 on DL20 (row 14), a 3.79-point increase over RankVicuna (row 13), the best single-call baseline model. For qualitative examples, see Appendix C.

Overall, our PSC approach consistently improves ordinary decoding and beats the maximum individual score across three runs (see scores in parentheses), yielding gains on 13 out of 16 model–dataset combinations (see PSC columns in rows 7–14). On average, RankVicuna, GPT-3.5, and GPT-4 see relative score increases of 0.4%, 2%, and 5% with PSC. Mixed results on RankVicuna likely result from its inherent robustness to positional bias, instilled by its training process that uses



(a) Single (GPT-3.5) on DL19 and DL20.

(b) Single (GPT-4) on DL19 and DL20.

Figure 4: Distribution of "reversions" after reranking. Blues are below the observed dataset average and reds above the average. For two input list positions $i \in [1, 20]$ and $j \in (i, 20]$, i indexes the rows and j the columns. For example, the cell at $(1, 2)$ is the reversion of the first two input items across the dataset. Note that highly saturated colors indicate over- and under-reversion *relative* to other pairs in the dataset rather than in the absolute sense.

random shuffling as part of data augmentation; thus, the shuffling step from PSC has less of an effect on the output variation.

The choice of the first-stage reranker has a clear impact, with SPLADE++ adding an average of 7.26 points over the corresponding BM25 models. In fact, reranking the top-20 SPLADE items (row 13) in a single call outperforms doing the top-100 (row 14) using a sliding call window. We conjecture that this results from imperfections in the RankGPT windowing algorithm, which shows especially for strong retrievers, where the top-20 already contains many relevant documents.

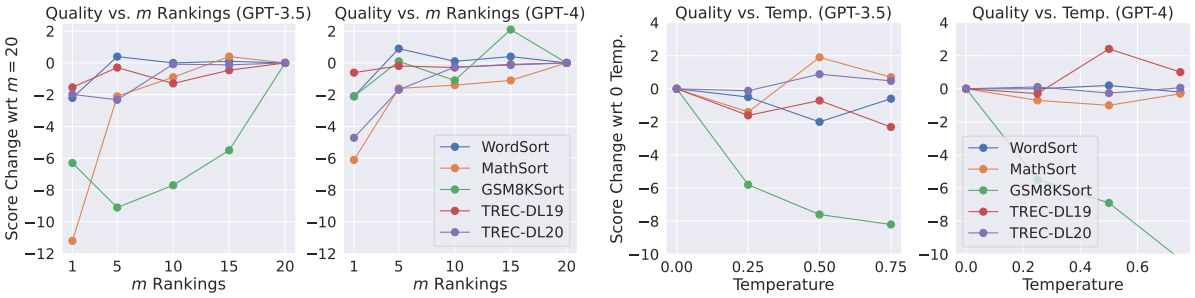
Finally, we note one particularly intriguing phenomenon: in the top-20 single-call setting, GPT-3.5 and GPT-4 have similar baseline quality without PSC (rows 8 and 9, first column in each group), but PSC boosts GPT-4 more than GPT-3.5 (row 9, second columns). As we explore in depth next, this possibly results from GPT-4 being more "equally biased" across the item positions and hence providing PSC more useful rankings for aggregation.

Positional bias analysis. We analyze how list order bias varies with the input positions on the "single" GPT models for BM25 (from Table 3, rows 8 and 9), which avoids confounds from RankGPT's window strategy. The design of our analysis is as follows, mirroring Section 2.2's notation: consider the item pair (X_a, X_b) with input list positions $(\pi_i(a), \pi_i(b))$, where $\pi_i(a) < \pi_i(b)$ for some random permutation π_i. If the output positions satisfy $\hat{\sigma}_i(a) > \hat{\sigma}_i(b)$ after reranking, we say the order is reversed, and we call the sum of reversed pairs per data point *"reversions."* In Figure 4, we visualize the distribution of reversions by input position pair, with $\pi_i(a)$ as the y-axis and $\pi_i(b)$ as the x-axis, whose positions range from 1–20 for each of the top-20 passages. For cross-model comparability, we normalize by dataset.

Under the null hypothesis of there being no positional bias, the distribution of reversions should be uniform because the input lists are randomly permuted, which severs any association between input order and output ranking. However, Figure 4 contradicts this. Prominently, the center of Figure 4a is redder than the edges, indicating that pairs with both items closer to the middle are reversed more often by GPT-3.5 than those at the beginning and the end of the input lists are. In Figure 4b, bottom areas are also deeper red than the top, showing that pairs with items at the end of the list are more frequently reversed by GPT-4 than pairs at the start.

Other subtle patterns emerge upon closer examination. First, in Figure 4a, a dark block appears after column 15, suggesting that GPT-3.5 does *not* focus well on items past the fifteenth. Second, the colors interleave in a grid pattern across both columns and rows—possibly an artifact of its pretraining. From this evidence, we conclude that different positional biases exist in reranking LLMs, varying by model and dataset.

The analysis also helps to explain our quality results. Comparing Figure 4a and 4b, we observe that GPT-4 generally reverses more pairs than GPT-3.5 and is closer to the optimal number of reversals, thus providing higher quality to the aggregated rankings. This may explain why PSC benefits GPT-4 (single) more than it does GPT-3.5 (single), i.e. row 9 vs. row 8 in Table 4. Similarly, both models tend to reverse more pairs on DL20 than on DL19, and results also indicate that PSC improves DL20 more than it does DL19.



(a) Quality vs. number of output rankings ($\rho = 0.17$).

(b) Quality vs. text generation temperature ($\rho = -0.078$).

Figure 5: Quality for all datasets for various aggregate sizes and temperatures. For output rankings, we use $m = 20$ as our frame of reference; for temperature, 0.0. In the subfigure captions, ρ denotes Spearman's rank correlation.

4 Sensitivity Analyses

In this section, we investigate and characterize each component of permutation self-consistency to justify our modeling choices.

4.1 Hyperparameter Studies

Output rankings. Throughout the paper, we espoused aggregating over $m = 20$ output rankings, but is more actually better? If, say, five outperformed twenty, we could decrease the number of parallel calls to the model, conceivably saving cost. To answer this question, we sweep the aggregate size between one and twenty across all datasets, plotting the resulting score differences from using the default twenty. We pick GPT-3.5 and GPT-4 as our target models, as they are used in all tasks.

We plot our results in Figure 5a. On both models, we find that output quality rapidly converges to that of using the full twenty, five being 67% as effective on average. The score averages increase monotonically with the number of rankings ($\rho = 0.17$), with GSM8KSort on GPT-3.5 as an outlier (left subplot), possibly because of output variance—the next study on sampling temperature shows that it is highly sensitive to randomness. We conclude that picking $m = 20$ output rankings is effective, though returns sharply diminish after 5–10.

Sampling temperature. Self-consistency (Wang et al., 2023b) uses temperature as their sampling strategy to produce different outputs to aggregate over, but it is ineffective for us, perhaps because listwise ranking does not admit multiple reasoning paths like chain-of-thought prompting does. To assess this rigorously, we vary the temperature between 0 and 0.75, following the original method's 0.5–0.7 (Wang et al., 2023b). For consistency, we use the same setup from before and fix $m = 20$.

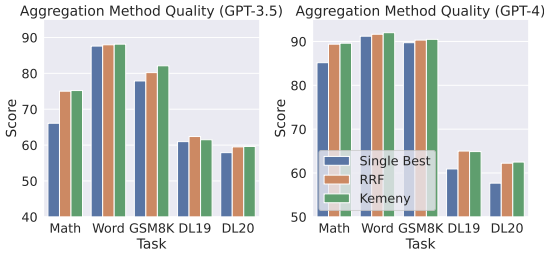


Figure 6: Scores for the alternative reciprocal rank fusion (RRF) and our Kemeny rank aggregation method.

We plot our results in Figure 5b. Temperature has little effect on the quality ($\rho = -0.078$), again with GSM8KSort as an outlier, where the extra randomness drastically hurts quality on both models. This sensitivity to randomness is also evident in Figure 3, where GSM8K has the widest interquartile range of the tasks. In conclusion, this evidence grounds our choice of not using temperature.

4.2 Rank Aggregation Comparison

Reciprocal rank fusion (RRF; Cormack et al., 2009) is a state-of-the-art alternative to our chosen Kemeny ranking method. It sorts items by the score

$$\text{RRFScore}(X_j) := \sum_{1 \le i \le m} \frac{1}{k + \hat{\sigma}_i(j)} \qquad (6)$$

for each item X_j, rankings $\hat{\sigma}_i$, and $k = 60$. RRF had been under our consideration, but we picked Kemeny ranking for its theoretical robustness and empirical effectiveness. Shown in Figure 6, Kemeny beats RRF ($p < 0.05$) on 8 out of 10 comparisons by a mean of 0.23 points; on average, RRF reaches only 93.5% of the boost that Kemeny does. Its only outperformance on DL19 possibly results from it being suited for information retrieval, its field of origin, but this may also be statistical noise. Overall, these results further support our decision to select Kemeny ranking for the aggregation step.

5 Related Work and Future Directions

The holistic direction of our work is in enhancing the ranking ability of large language models. Along a similar vein, contrast-consistent ranking (Stoehr et al., 2023) proposes to train order-unaware probes on the latent vectors of large language models for detecting nondirectional rank consistency. Their evaluation reveals the ranking direction of test examples to the models, deviating from standard practices, as their purpose is not to increase ranking quality but rather to detect consistency. Another related work is Hou et al. (2023), which uses a different rank aggregation algorithm from ours. In contrast to their heuristic bootstrapping method (i.e., Borda count) of summing up the ranks of each ranking, our approach is theoretically optimal in that it finds the best central ranking to all individual rankings in terms of the tau distance.

The specific empirical tasks in this paper have also seen recent progress. For passage ranking using language models, BERT-based (Devlin et al., 2019; Nogueira et al., 2020) and T5-tuned (Zhuang et al., 2023; Raffel et al., 2020) approaches represent the earliest language models for passage ranking. RankGPT (Sun et al., 2023) and LRL (Ma et al., 2023b) spearheaded much of the post-ChatGPT work, beating the supervised state of the art with an *unsupervised* LLM for the first time. Along a non-listwise direction, PRP (Qin et al., 2023) is a pairwise method leveraging open-source large language models comparing two items at a time, as reported in Table 4. One possible future work is to reformulate our PSC method to be differentiable, enabling training-time application in LLMs such as RankVicuna (Pradeep et al., 2023).

Our sorting tasks for LLMs have had attention as well, mostly in the context of evaluation, with BIG-Bench (Suzgun et al., 2022; bench authors, 2023), an LLM benchmark, providing more than 200 distinct tasks, including one in alphabetical ordering (`word_sorting`), which we enlarge and expand on in WordSort. Stoehr et al. (2023) also constructed fact-based synthetic sorting datasets for listwise ranking, but they are private and hence noncomparable. In the future, PSC can be applied to any list-oriented ranking task involving LLMs. Examples include using LLMs for evaluation (Wang et al., 2023a) and annotating human feedback judgments with language models. Additionally, PSC is applicable at training time, such as denoising weakly labeled training sets generated by teacher models, shown to be crucial to the success of listwise-ranking LLMs (Pradeep et al., 2023).

We are not the first to establish positional biases in LLMs. Lu et al. (2022) are among the earliest to relate prompt order to the quality of in-context learning. The main difference in setup is that they assume the presence of a training set, whereas we do not, which especially matters for passage ranking, as many tasks only have evaluation sets. Recently, Liu et al. (2023) and Wang et al. (2023a) characterized positional bias in the context of list-oriented tasks, such as question answering and response evaluation. However, we are to our knowledge the first to characterize the position biases of passage-ranking LLMs with respect to pairwise item positions, and our work also proposes a correction technique. Moreover, Pezeshkpour and Hruschka (2023) and Li et al. (2023) apply prompting-based techniques for mitigating positional bias. Prompting is not mutually exclusive of our PSC, and it could be complementary.

Lastly, our paper is connected to all the meta-algorithms for improving LLM generation. As a pertinent example, Lu et al. (2022) study prompt order on in-context learning classification tasks, proposing an entropy-based statistic over development sets to find performant permutations of few-shot examples. Aggarwal et al. (2023) make self-consistency more efficient, halting the procedure when enough samples have been collected. To keep our method in its simplest form, as self-consistency had not been applied to listwise ranking to begin with, we based our design on the original approach (Wang et al., 2023b).

6 Conclusions

We introduce a novel decoding method to improve the ranking ability of black-box LLMs by mitigating potential sensitivities and biases to list item order. We intervene on prompt list order to produce multiple rankings then return an aggregated statistic as the prediction, which has less association with list order. Theoretically, we prove the robustness of our method to arbitrary, fixed noise distributions. Empirically, our method consistently improves upon ordinary decoding on all 15 of our sorting model–dataset combinations and 13 out of 16 of our passage reranking ones. Finally, our sensitivity analyses justify our design choices of 20 output rankings, zero sampling temperature, and the Kemeny ranking method.

Limitations

We share limitations with those of the original self-consistency paper (Wang et al., 2023b). We use multiple LLM calls, potentially to a commercial LLM, which would raise financial cost. Thus, practical applications may require careful weighing of quality gain against elevated expense. Nevertheless, a few calls already help, and returns rapidly diminish past 5–10 calls. We note that our method does not in practice increase latency by much, since all calls can be parallelized, and aggregation time does not rise with the number of samples. For further discussion, see Appendix D.3.

Another limitation is that GPT-3.5 and GPT-4 are proprietary models lacking official documentation of its internals. We acknowledge that this is an ongoing issue in the natural language processing literature as of 2023, with many publications relying on the continued existence of these endpoints. To partially alleviate this, we have run experiments on the open-source Mistral (Jiang et al., 2023), Zephyr (Tunstall et al., 2023), LLaMA 2 (Touvron et al., 2023), and RankVicuna (Pradeep et al., 2023) models where possible.

Finally, our study is intentionally restricted to automated evaluation in an academic setting. Kendall's tau and nDCG@10, while standard metrics in evaluating ranking systems, do not exactly capture human preferences. It remains to be determined how effective permutation self-consistency is for, say, an in-production web search engine or recommendation system.

References

Pranjal Aggarwal, Aman Madaan, Yiming Yang, et al. 2023. Let's sample step by step: Adaptive-consistency for efficient reasoning with LLMs. *arXiv:2305.11860*.

Alnur Ali and Marina Meilă. 2012. Experiments with Kemeny ranking: What works when? *Mathematical Social Sciences*.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. 2016. MS MARCO: A human generated machine reading comprehension dataset. *arXiv:1611.09268*.

BIG bench authors. 2023. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*.

Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, and others. 2023. Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv:2110.14168*.

Vincent Conitzer, Andrew Davenport, and Jayant Kalagnanam. 2006. Improved bounds for computing Kemeny rankings. In *Proceedings of the 21st National Conference on Artificial Intelligence*.

Gordon V. Cormack, Charles Clarke, and Stefan Buettcher. 2009. Reciprocal rank fusion outperforms Condorcet and individual rank learning methods. In *SIGIR*.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2021. Overview of the TREC 2020 deep learning track. *arXiv:2102.07662*.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. 2020. Overview of the TREC 2019 deep learning track. *arXiv:2003.07820*.

Tri Dao. 2023. FlashAttention-2: Faster attention with better parallelism and work partitioning. *arXiv:2307.08691*.

Tri Dao, Dan Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. *Advances in Neural Information Processing Systems*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL: HLT*.

Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE v2: Sparse lexical and expansion model for information retrieval. *arXiv:2109.10086*.

Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. 2023. Large language models are zero-shot rankers for recommender systems. In *ECIR*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7B. *arXiv:2310.06825*.

John G. Kemeny. 1959. Mathematics without numbers. *Daedalus*.

Maurice George Kendall. 1948. Rank correlation methods.

Ruosen Li, Teerth Patel, and Xinya Du. 2023. PRD: Peer rank and discussion improve large language model based evaluations. *arXiv:2307.02762*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2023. Lost in the middle: How language models use long contexts. *arXiv:2307.03172*.

Yao Lu, Max Bartolo, Alastair Moore, Sebastian Riedel, and Pontus Stenetorp. 2022. Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity. In *ACL*.

Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. 2023a. Fine-tuning LLaMA for multi-stage text retrieval. *arXiv:2310.08319*.

Xueguang Ma, Xinyu Zhang, Ronak Pradeep, and Jimmy Lin. 2023b. Zero-shot listwise document reranking with a large language model. *arXiv:2305.02156*.

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document ranking with a pre-trained sequence-to-sequence model. In *Findings of ACL: EMNLP 2020*.

Pouya Pezeshkpour and Estevam Hruschka. 2023. Large language models sensitivity to the order of options in multiple-choice questions. *arXiv:2308.11483*.

Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. 2023. RankVicuna: Zero-shot listwise document reranking with open-source large language models. *arXiv:2309.15088*.

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, Xuanhui Wang, et al. 2023. Large language models are effective text rankers with pairwise ranking prompting. *arXiv:2306.17563*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*.

Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*.

Niklas Stoehr, Pengxiang Cheng, Jing Wang, Daniel Preotiuc-Pietro, and Rajarshi Bhowmik. 2023. Unsupervised contrast-consistent ranking with language models. *arXiv:2309.06991*.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT good at search? investigating large language models as re-ranking agents. In *EMNLP*.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, et al. 2022. Challenging BIG-Bench tasks and whether chain-of-thought can solve them. *arXiv:2210.09261*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv:2307.09288*.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, et al. 2023. Zephyr: Direct distillation of lm alignment. *arXiv:2310.16944*.

Peiyi Wang, Lei Li, Liang Chen, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. 2023a. Large language models are not fair evaluators. *arXiv:2305.17926*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-consistency improves chain of thought reasoning in language models. In *ICLR*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V. Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv:2303.18223*.

Honglei Zhuang, Zhen Qin, Rolf Jagerman, Kai Hui, Ji Ma, Jing Lu, Jianmo Ni, Xuanhui Wang, and Michael Bendersky. 2023. RankT5: Fine-tuning T5 for text ranking with ranking losses. In *SIGIR*.

A Proofs of Propositions

Proposition A.1 (2.1)**.** *Let there be a true ranking σ and a sequence of i.i.d. uniformly noisy rankings $\hat{\boldsymbol{\sigma}} := \{\hat{\sigma}_i\}_{i=1}^m$. Suppose each noisy ranking $\hat{\sigma}_k$ has a uniformly random, nonempty concordant subset S'_k with σ, and the remaining rank elements not in S'_k represent a random permutation. Then the Kemeny–Young ranking $\bar{\sigma}$ of $\hat{\boldsymbol{\sigma}}$ converges in probability to σ, i.e., it is a consistent estimator.*

Proof. Our strategy is to upper-bound the probability that the number of discordant pairs between the predicted rankings and the true ranking is greater than the number of concordant pairs, then show that this upper bound approaches zero with enough samples. Since the Kemeny-optimal ranking is defined

as the ranking that exactly minimizes the difference between these discordant and concordant pairs, the probability that the Kemeny ranking is equivalent to the true ranking shares this upper bound.

Let A_{ij} be the event that the sum of discordant pairs indexed by i and j between $\hat{\boldsymbol{\sigma}}$ and σ is greater than the concordant ones, i.e.,

$$\begin{aligned} A_{ij} &:= \sum_{k=1}^{m} \text{disc}_{ij}(\sigma, \hat{\sigma}_k) > \sum_{k=1}^{m} \text{conc}_{ij}(\sigma, \hat{\sigma}_k) \\ &= \sum_{k=1}^{m} \text{disc}_{ij}(\sigma, \hat{\sigma}_k) > m - \sum_{k=1}^{m} \text{disc}_{ij}(\sigma, \hat{\sigma}_k) \\ &= \sum_{k=1}^{m} \text{disc}_{ij}(\sigma, \hat{\sigma}_k) > \frac{m}{2}, \end{aligned}$$

with convenience functions $\text{disc}_{ij}(\sigma_1, \sigma_2) := \mathbb{I}((\sigma_1(i) > \sigma_2(j) \wedge \sigma_1(i) < \sigma_2(j)) \vee (\sigma_1(i) < \sigma_2(j) \wedge \sigma_1(i) > \sigma_2(j)))$ and $\text{conc}_{ij} := 1 - \text{disc}_{ij}(\sigma_1, \sigma_2)$ indicating pair discordance and concordance according to the Kendall tau criterion. The LHS of the event also defines a sum of independent random variables (r.v.), each a Bernoulli distribution

$$X_k = \text{disc}_{ij}(\sigma, \hat{\sigma}_k) \sim \text{Bernoulli}(p_k). \tag{7}$$

This is evident because each summand is a binary outcome, which is independent based on the given premise that $\hat{\boldsymbol{\sigma}}$ and concordant subsets $\{S'_k\}_{k=1}^m$ are each independent. The probability of A_{ij} can be equivalently stated as

$$\mathbb{P}(A_{ij}) = \mathbb{P}(\sum_{k=1}^{m} X_k > \frac{m}{2}). \tag{8}$$

We upper-bound the RHS. First, since S'_k is non-empty and $\hat{\sigma}_k$'s elements not in S'_k are uniformly random (as given by the premise), the chance of drawing a discordant ranking is $p_k \leq \frac{1}{2} - \delta$ for some $\delta > 0$. Thus,

$$\begin{aligned} \mathbb{P}(\sum_{k=1}^{m} X_k > \frac{m}{2}) &\leq \mathbb{P}(\sum_{k=1}^{m} X > \frac{m}{2}) && (9) \\ &= \mathbb{P}(\frac{1}{m}\sum_{k=1}^{m} X > \frac{1}{2}), && (10) \end{aligned}$$

where $X \sim \text{Bernoulli}(\frac{1}{2} - \delta)$. By Hoeffding's inequality, we have for all $\epsilon > 0$

$$\mathbb{P}\left(\frac{1}{m}\sum_{k=1}^{m} X - (\frac{1}{2} - \delta) > \epsilon\right) \leq \exp(-2m\epsilon^2).$$

Let $\epsilon = \delta$. Then

$$\begin{aligned} &\mathbb{P}\left(\frac{1}{m}\sum_{k=1}^{m} X - (\frac{1}{2} - \delta) > \delta\right) && (11) \\ &= \mathbb{P}\left(\frac{1}{m}\sum_{k=1}^{m} X - \frac{1}{2} > 0\right) && (12) \\ &= \mathbb{P}\left(\frac{1}{m}\sum_{k=1}^{m} X > \frac{1}{2}\right) && (13) \\ &= \mathbb{P}(A_{ij}) && (14) \\ &\leq \exp(-2m\delta^2). && (15) \end{aligned}$$

We now consider the probability of any A_{ij}, i.e., the probability that the sum of discordant pairs indexed by *any* i and j between $\hat{\boldsymbol{\sigma}}$ and σ is greater than the concordant ones. By the union bound,

$$\begin{aligned} \mathbb{P}(\bigcup_{i<j} A_{ij}) &\leq \sum_{i<j} \mathbb{P}(A_{ij}) && (16) \\ &\leq \binom{n}{2} \exp(-2m\delta^2) && (17) \\ &\leq n^2 \exp(-2m\delta^2). && (18) \end{aligned}$$

Taking $m \to \infty$, the RHS $= 0$. Since the Kemeny-optimal ranking always chooses the ranking that minimizes pairwise discordance (picking any other ranking would increase the Kendall tau distance, a contradiction with the definition of Kemeny optimality), for $m \to \infty$ we recover the true ranking with probability 1, completing our proof that it is a consistent estimator. □

Proposition A.2 (2.2)**.** *Let there be a true ranking σ and a distribution of noisy rankings $\mathbb{P}(\sigma_{noise})$, where $\sigma_{noise} \circ \pi$ always has a uniform, non-empty concordant subset S with σ for any input ranking π, and the elements not in S are uniformly random. Then the permutation self-consistency procedure is a consistent estimator of σ when applied to the input π and the "LLM" characterized by $\mathbb{P}(\sigma_{noise})$.*

Proof. Our technique is to show that the premises of Proposition 2.2 can be transformed to those of 2.1, which we have a proof for. Let π be drawn uniformly at random from the sample space of all permutations, Ω, as in the first step of the permutation self-consistency procedure. From the premise of both the concordant subset S of $\sigma_{\text{noise}} \circ \pi$ and its complement S^C being uniformly random, letting $\hat{\boldsymbol{\sigma}}$ be realizations of $\sigma_{\text{noise}} \circ \pi$ fulfills the premise for Proposition 2.1. The rest of our proof follows from that of 2.1. □

1. MathSort: Sort ten arithmetic expressions by value.

<User> Sort the following expressions from smallest to largest: `3 / 5, 2 - 9, 6 * 5, 2 * 1, 3 / 1, 9 * 9, 1 - 9, 9 + 8, 3 / 5, 1 / 9`. The output format should be a comma-separated list containing the exact expressions; do not reduce them. Only respond with the results; do not say any word or explain.

2. WordSort: Order ten words alphabetically.

<User> Order these words alphabetically: `aaron`, `roam`, `aardvark`, `nexus`, [...]. The output format should [...]

3. GSM8KSort: Unscramble sentences from GSM8K.

<User> Order the scrambled sentences logically:
- `She took 1 hour to walk the first 4 miles` [...]
- `Marissa is hiking a 12-mile trail.`
- `If she wants her average speed to be 4` [...]

The output format should have each sentence on a new line. Only respond with the results; do not say any [...]

Table 5: Full prompts for our three sorting tasks. "<User>" is a model-specific prefix token qualifying the subsequent message as belonging to the user for instruction prompting.

B Detailed Experimental Setup

B.1 Computational Environment

We conducted the experiments on a machine running Ubuntu 22.04 with two Nvidia A6000 GPUs, an AMD Epyc Milan 7B13 CPU, and 256GB of ECC RAM. Our most relevant software frameworks included PyTorch 2.1.0, Transformers 4.36.1, PuLP 2.7.0, and CUDA 12.2. Where possible, we used FlashAttention v2 (Dao, 2023; Dao et al., 2022) and BF16 to accelerate the LLMs.

B.2 Sorting Tasks

Table 5 lists the full prompts used in our sorting tasks. To extract the rankings, we examined the outputs and wrote regular expressions; all the models capably generated well-formed, extractable text, in line with the claims in their papers (Tunstall et al., 2023; Jiang et al., 2023; Touvron et al., 2023). All prompts fit in a context size of 4096 tokens.

Dataset settings. We made a few further considerations in designing WordSort and MathSort. To add difficulty to word sorting, for each example we randomly mixed five consecutively ordered words in the English language with five randomly picked ones, e.g., mixing "aardvark, aaron, abacus ..." with "dog, cat, shrew ..." On MathSort, we ensured that all expressions evaluated to unique values within an example (of 10 expressions). None of our lists for any task were duplicates.

RankVicuna: Prompt from Pradeep et al. (2023).

<User> I will provide you with {num} passages, each indicated by a numerical identifier []. Rank the passages based on their relevance to the search query: {query}.

[1] {passage 1}
[2] {passage 2}
...
[{num}] {passage {num}}

Search Query: {query}.
Rank the {num} passages above based on their relevance to the search query. All the passages should be included and listed using identifiers, in descending order of relevance. The output format should be [] > [], e.g., [4] > [2]. Only respond with the ranking results, do not say any word or explain.

RankGPT: Prompt from Sun et al. (2023).

<System> You are RankGPT, an intelligent assistant that can rank passages based on their relevancy to the query.

<User> I will provide you with {num} passages, each indicated by number identifier [].
Rank the passages based on their relevance to query: {query}.

[1] {passage 1}
[2] {passage 2}
...
[{num}] {passage {num}}

Search Query: {query}.
Rank the {num} passages above based on their relevance to the search query. The passages should be listed in descending order using identifiers. The most relevant passages should be listed first. The output format should be [] > [], e.g., [1] > [2]. Only response the ranking results, do not say any word or explain.

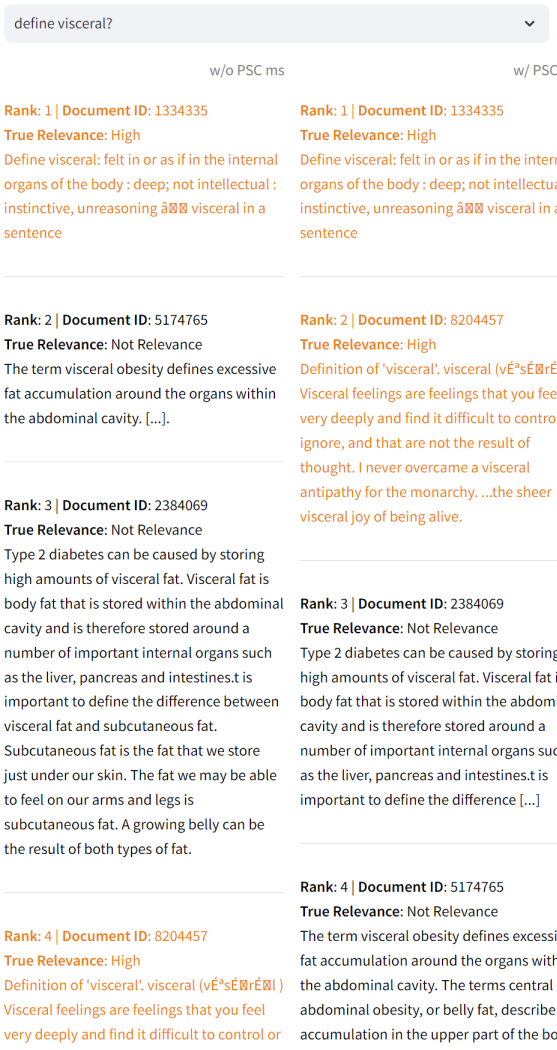
Table 6: Full prompts for our passage reranking task. "<User>" and "<System>" are model-specific prefix tokens denoting the user and system roles. Nuances between RankVicuna and RankGPT include grammatical changes and no system prompt.

B.3 Passage Reranking Task

Table 6 lists the full prompts for our passage reranking task, following prior art precisely (Sun et al., 2023; Pradeep et al., 2023). We used the same output extraction procedure from the official codebases as well, ensuring a faithful comparison.

For our OpenAI GPT endpoints, we deployed GPT-3.5 Turbo (version 0613) and GPT-4 on Azure. In total, at the current public price of $0.002 and $0.03 per one thousand tokens,[1] we estimate a cost of $100–200 USD to reproduce the GPT passage ranking results in their entirety, with GPT-4 consuming most of it.

[1] https://openai.com/pricing



(a) Results reranked by GPT-3.5. Both PSC and conventional inference rank the first document the same, but PSC correctly ranks document #8204457 higher (second vs. fourth).

(b) Results reranked by GPT-4. Compared to the previous example, PSC results in no difference.

Figure 7: The DL19 query "define visceral?" with relevant documents reranked without PSC on the left and with PSC on the right of each subfigure.

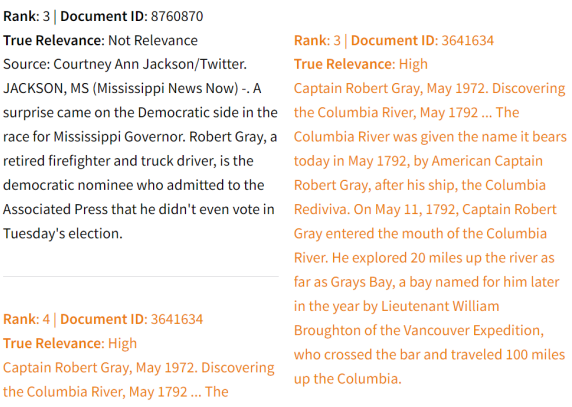


Figure 8: The DL19 query "who is robert gray" with relevant documents reranked without PSC on the left and with PSC on the right, with GPT-4 as the model.

C Qualitative Examples

We present qualitative examples of our approach on DL19 in Figures 7a, 7b, and 8. In Figures 7a and 7b, we compare the outputs of GPT-3.5 and GPT-4 with PSC, fixing the query to "define visceral?" We find that PSC improves GPT-3.5 but not GPT-4, since GPT-4's original output is already correct, providing visual evidence for why PSC attains more gains on GPT-4 than on GPT-3.5. In Figure 8, GPT-4 with PSC ranks the third document (#3641634) correctly higher (right) than GPT-4 without PSC (left). In summary, these illustrations suggest that the quantitative improvements of PSC are not merely illusory.

First Stage	Top-k	Method	TREC-DL19		TREC-DL20	
			Original	Our PSC	Original	Our PSC
BM25	20	(1) Single (GPT-3.5, Reversed)	55.92	**62.88**	52.66	**59.59**
	20	(2) Single (GPT-4, Reversed)	64.04	**65.60**	60.20	**62.27**
	100	(3) RankGPT (GPT-3.5, Reversed)	56.76	**57.32**	51.03	**55.73**
	100	(4) RankGPT (GPT-4, Reversed)	67.83	**69.63**	64.92	**65.89**

Table 7: nDCG@10 results on DL19 and 20.

Method	MATHSORT	WORDSORT	GSM8KSORT
GPT-3.5 (PRP)	46.7	82.2	64.0
GPT-4 (PRP)	73.3	83.9	79.9
GPT-3.5 (PSC)	75.2	88.1	88.4
GPT-4 (PSC)	**89.6**	**92.0**	**90.5**

Table 8: Pairwise ranking prompting versus permutation self-consistency on the sorting tasks.

Method	MATH	WORD	GSM8K	DL19	DL20
GPT-3.5 (Orig.)	64.0	85.9	82.1	68.00	62.08
GPT-3.5 (Borda)	74.6	87.9	88.1	70.09	62.54
GPT-3.5 (Our PSC)	**75.2**	**88.1**	**88.4**	**70.77**	**62.70**
GPT-4 (Orig.)	83.5	89.9	88.4	75.00	70.36
GPT-4 (Borda)	89.2	91.5	90.4	75.23	70.62
GPT-4 (Our PSC)	**89.6**	**92.0**	**90.5**	**75.66**	**71.00**

Table 9: Comparisons to using bootstrapping/Borda count as the aggregation algorithm.

D Supplementary Results and Discussion

During the peer review process of this paper, our reviewers helpfully suggested experiments to further bolster the rigor of our claims. We explicitly include most of them here, with the remaining feedback incorporated into the related work section.

D.1 Sorting Tasks

In Table 8, we demonstrate that our PSC approach outperforms PRP-10 by 15 points in Kendall's tau on average, with higher gains on GPT-3.5. We chose PRP-10 because it most closely matches ours in computation time. Overall, these findings are in line with our conclusions on the passage reranking task, as shown in Table 4.

D.2 Passage Reranking Task

We additionally conducted experiments on reversing input orders for passage reranking. Note that it is inapplicable to sorting because the underlying dataset is unordered, so reversing the input would not affect the results. Shown in Table 7, our PSC method improves the result by an average of 3.2 points; thus, it successfully mitigates position bias regardless of the order used in the sliding window.

Finally, we show that PSC outperforms the bootstrapping technique (or "Borda count," as the rank aggregation literature calls it) from Hou et al. (2023). Presented in Table 9, our method consistently outperforms bootstrapping on GPT-3.5 and GPT-4, possibly because of Kemeny ranking's theoretical optimality. The gains are roughly equal between GPT-3.5 (0.39 average point increase) and GPT-4 (0.42 points). We conclude that the choice of rank aggregation algorithm matters.

D.3 Computational Burden

Finally, we discuss the difference in time and computational cost of the proposed method compared to other baselines. We concede computation time is a general limitation of many self-consistency-style works, as we acknowledge in our limitations section. Our advantage is that PSC is embarrassingly parallel and scales horizontally with ease, e.g., for a choice of five repetitions, spinning up five instances will be roughly equivalent to the original baseline of one. Furthermore, our Kemeny-optimal aggregation method is virtually instantaneous for practical sample sizes (less than 0.05 CPU seconds). This contrasts with methods such as PRP that necessitate, say, 20–200 sequential calls for a list size of 10 rather than our fully parallelizable 5–20.

Thus, even though in theory our method is asymptotically linear, in practice the big-O constant can be made "small" by horizontal scaling, assuming the presence of parallel computing. As a rough quantitative comparison, our experiments run 20 parallel calls to multiple deployments of GPT-3.5 and GPT-4, incurring a running time of no more than 25% (in addition to) a single call.