Batched Self-Consistency Improves LLM Relevance Assessment and Ranking

Anonymous ACL submission

Abstract

Given some information need, Large Language Models (LLMs) are increasingly used for candidate text relevance assessment, typically using a one-by-one pointwise (PW) strategy where each LLM call evaluates one candidate at a time. Meanwhile, it has been shown that LLM performance can be improved through self-consistency: prompting the LLM to do the same task multiple times (possibly in perturbed ways) and then aggregating the responses. To take advantage of self-consistency, we hypothesize that batched PW strategies, where multiple passages are judged in one LLM call, are better suited than one-by-one PW methods since a larger input context can induce more diverse LLM sampling across self-consistency calls. We first propose several candidate batching strategies to create prompt diversity across self-consistency calls through subset reselection and permutation. We then test our batched PW methods on relevance assessment and ranking tasks against one-by-one PW and listwise LLM ranking baselines with and without self-consistency, using three passage retrieval datasets and GPT-4o, Claude Sonnet 3, and Amazon Nova Pro. We find that batched PW methods outperform all baselines, and show that batching can greatly amplify the positive effects of self-consistency. For instance, on our legal search dataset, GPT-4o one-by-one PW ranking NDCG@10 improves only from 44.9% to 46.8% without self-consistency vs. with it, while batched PW ranking improves from 43.8% to 51.3%, respectively.

1 Introduction

LLM relevance assessment and ranking have become two foundational tasks for agentic artificial intelligence (AI) and retrieval-augmented generation (RAG) systems (Wang et al., 2024). Given an information need, existing LLM candidate relevance assessment methods typically use a one-by-one pointwise (PW) strategy where each LLM call evaluates one candidate at a time (Thomas et al., 2024; Upadhyay et al., 2024). While such methods can avoid positional biases due to candidate order, they prevent information sharing between multiple candidates and can be computationally expensive, requiring a separate LLM call for each candidate. For LLM ranking, several techniques exist including PW, listwise (LW) (Tang et al., 2024; Ma et al., 2023; Sun et al., 2023), and pairwise (Qin et al., 2024; Liusie et al., 2024) approaches, but only PW methods can rank *and* generate absolute relevance judgments.

Meanwhile, a simple technique for improving LLM output generation quality has emerged called *self-consistency* (Wang et al., 2023) which prompts the LLM to do the same task multiple times (possibly with prompt perturbations) and then aggregates the outputs – leveraging the variability in LLM decoding. Motivated by these observations, we propose that batched PW scoring, where multiple passages receive relevance scores in one LLM call, is especially well suited for self-consistency methods. Compared to one-by-one PW appraoches, we hypothesize that the larger context provided by batched PW methods can induce more helpful variation between LLM self-consistency calls, as well as benefit from diverse prompts created with various candidate subsets and permutations.

We evaluate these batched PW methods against one-by-one PW and LW techniques with and without self-consistency using GPT-4o, Claude Sonnet 3, and Amazon Nova Pro on passage relevance assessment and ranking tasks from three datsets. Our contributions include:

- We propose several batching strategies (Figure 1) to diversify LLM self-consistency prompts using various candidate subsets and permutations.

- We show that batching can greatly amplify the positive effects of self-consistency. For

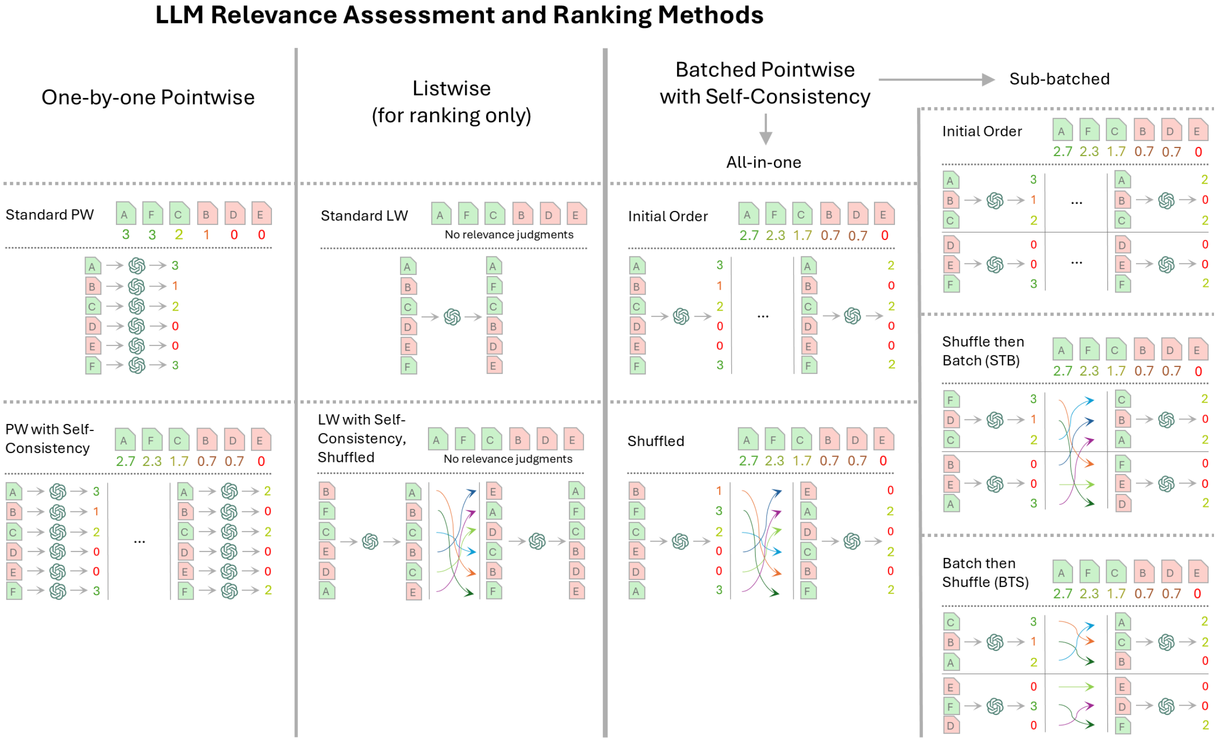

Figure 1: One-by-one PW methods (first pane) evaluate each candidate in a separate LLM call. LW methods (second pane) prompt the LLM to rerank the candidate list but do not produce relevance judgments. All-in-one batched PW methods (third pane) evaluate *all* candidates in each call while sub-batched PW methods (right pane) select a subset. Self-consistency calls can repeat identical LLM calls, or use various candidate permutations (LW and batched PW only), and/or different candidate subsets (sub-batched PW only).

instance, on legal search, GPT-4o one-by-one PW ranking NDCG@10 improves from 44.9% without self-consistency to 46.8% with self-consistency – while the best batched PW ranking method improves from 43.8% to 51.3%, respectively.

- We find that large batches can introduce harmful position biases in all three LLMs, but these can be successfully mitigated by using smaller batches, shuffling, and self-consistency.
- We observe that batched self-consistency PW methods outperform all LW and one-by-one PW methods with self-consistency.

2 Related Work

We briefly review existing LLM and non-LLM text relevance assessment and ranking techniques as well as emerging work on LLM self-consistency.

2.1 Text Relevance Assessment and Ranking

Given a query q, both non-generative and generative approaches are widely used to rank or score candidate text spans in some collection $\{p_1, \cdots, p_D\}$ based on relevance to q. Non-neural, sparse methods such as TF-IDF (Salton et al., 1975) and its probabilistic variant BM25 (Robertson et al., 2009) rely on syntactic token matches, limiting their ability to capture semantic similarity. Encoder-only LLM methods broadly include bi-encoders (Izacard et al., 2021; Gao and Callan, 2021) which score using a similarity function (e.g. cosine similarity) between separately embedded queries and passages, and cross-encoders (Nogueira and Cho, 2019; Zhuang et al., 2023) which jointly embed queries and passages to predict a relevance score. While these foundational methods are critical for retrieving initial candidate lists from large corpora, none of them are able to benefit from self-consistency since they do not use stochastic decoding or prompting.

Generative LLM-based methods typically consist of PW, pairwise, and LW strategies. Standard one-by-one PW and LW techniques (with and without self consistency) are baselines discussed further in sections 3.1.1 and 3.1.2, respectively. Pairwise rankers (Qin et al., 2024; Liu et al., 2024) ask an LLM which passage out of a pair (p_i, p_j) is more relevant, but need a quadratic number of LLM calls relative to the number of candidates. There are also bubble-sort sliding-window based LW LLM rankers (Ma et al., 2023; Sun et al., 2023) requiring *sequential* LLM calls, but we instead focus on fully parallelizable methods.

2.2 LLM Self-Consistency

LLM self-consistency (Wang et al., 2023) involves prompting an LLM to do the same task multiple times (potentially with perturbed prompts) and then aggregating the outputs, improving response quality by leveraging the stochasticity in LLM decoding. Multiple variations exist (Li et al., 2024), including using perturbations of few-shot examples to diversify prompts (Lu et al., 2022), and using heterogeneous LLM ensembles (Wan et al., 2024). Recent work (Tang et al., 2024; Hou et al., 2024) reported that self-consistency can improve LW LLM ranking, as discussed further in Section 3.1.2

3 Methodology

Figure 1 outlines the methods we study for LLM relevance assessment and ranking, all of which take as input a query q and an initial list of D candidate passages $L^q = [p_1, ..., p_D]$. All methods rerank the initial list, but only PW methods generate relevance predictions.

PW Relevance Labels: In all PW methods, each LLM call generates a relevance score $s_{q,p} \in \mathbb{R}$ between q and each passage $p \in L^q$. For ranking, passages are sorted in descending score order with ties broken using the order of the initial list L^q. We use the following 0-3 scale from the UMBRELLA open-source Bing prompt (Upadhyay et al., 2024), with our full prompt shown in Appendix C:

- **3:** The passage is dedicated to the query and contains the exact answer.
- **2:** The passage has some answer for the query, but the answer may be a bit unclear, or hidden amongst extraneous information.
- **1:** The passage seems related to the query but does not answer it.
- **0:** The passage has nothing to do with the query.

Self-Consistency All our self-consistency methods include each passage in exactly m LLM calls. In our PW self-consistency methods, each passage thus receives m scores $\{s^1_{q,p}, \cdots, s^m_{q,p}\}$, which are aggregated into a final score $s_{q,p}$ by taking the mean.[1] Similarly, our LW methods aggregate m output lists by minimizing Kendall-Tau distance, as described further in Sec. 3.1.2.

3.1 Baselines

3.1.1 One-by-one Pointwise Methods

The first pane of Figure 1 shows one-by-one PW methods (Thomas et al., 2024; Upadhyay et al., 2024; Törnberg, 2024), where each passage is scored against the query in a separate LLM call. While evaluating one candidate at a time avoids inducing candidate order position biases, it also prevents the LLM from seeing potentially helpful context in other passages. Further, each passage score requires a separate LLM call, which typically leads to a very large number of calls – especially with self-consistency.

3.1.2 Listwise Ranking Methods

LW ranking methods (Ma et al., 2023; Tang et al., 2024; Hou et al., 2024) are shown in the second pane of Figure 1.

Standard LW: Standard LW ranking instructs an LLM to rerank an input passage list in order of relevance with respect to q, with our LW prompt shown in Appendix C. While the LLM sees all available passage context during a single inference, no absolute relevance judgments are produced.

LW with Self-Consistency: LW ranking with self-consistency (Tang et al., 2024; Hou et al., 2024) involves m reranking calls followed by a rank aggregation of m output passage lists, which can be done by minimizing the Kendall-Tau distance with the lists. Multiple exact and approximate aggregation techniques exist, with our experiments using the exact Kemeny rank aggregation linear program (LP) of Tang et al.[2] We test two LW self-consistency variants:

1. **Initial Order:** Each of m LLM calls is identical and maintains the initial input list order.
2. **Shuffled:** The passage list is fully shuffled before each LLM call.

[1] For ranking with integer 0-3 scores, mean aggregation reduces the number of ties compared to majority voting.

[2] https://github.com/castorini/perm-sc

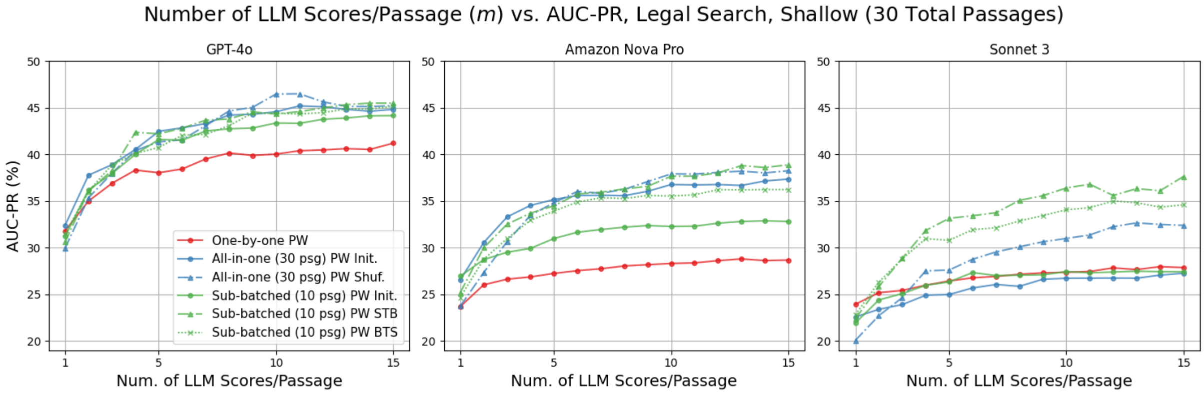

Figure 2: Effect of increasing self-consistency calls/passage (m) on PW relevance assessment quality (Legal Search, Shallow). For all LLMs at $m = 1$ (no self-consistency), one-by-one PW is competitive, but by $m = 15$ it underperforms batched PW by 5-10% AUC-PR. This shows that batching amplifies the benefits of self-consistency, likely due to broader input contexts.

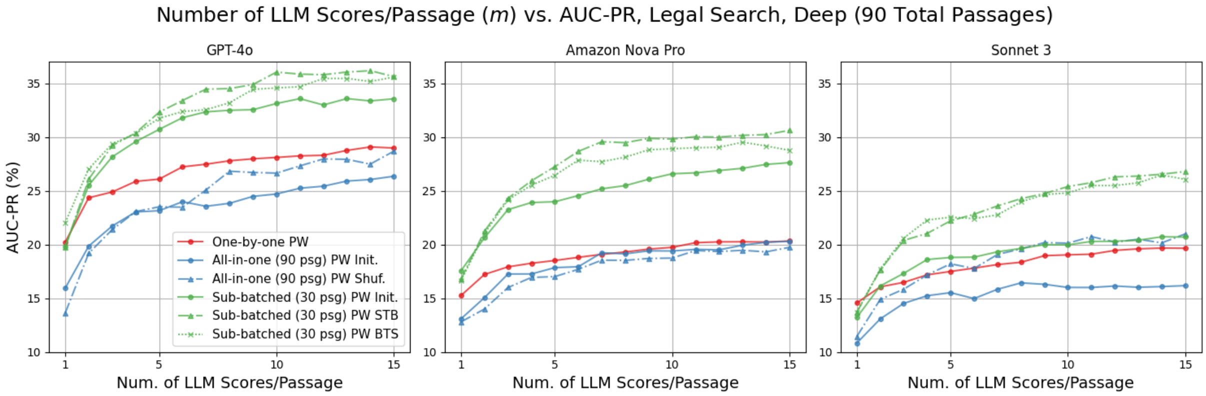

Figure 3: Legal Search, Deep (90 Total Psgs): effect of increasing m on relevance assessment quality. The sub-batched methods (30 psgs/batch) perform very well, with the shuffling variants (STB and BTS) doing best. The large batch (90 psgs/batch) all-in-one methods perform poorly for this range of m, addressed further in RQ2.

3.2 Batched PW Methods

The right half of Figure 1 shows batched PW methods in which multiple passages are jointly scored in each LLM call. This allows an LLM to see context from multiple passages at the same time and reduces the total number of LLM calls required compared to one-by-one methods. We study the effects of diversifying passage subsets and permutations through several batching strategies, including all-in-one batching (c.f. Sec. 3.2.1) and sub-batching (c.f. Sec. 3.2.2).

3.2.1 All-in-one

All-in-one PW methods prompt the LLM to score all passages in a single batch,[3] maximizing the context available to the model but also presenting it with the most complex task. We test two passage ordering strategies:

1. **Initial Order:** The initial list order is kept, giving m identical self-consistency calls.

2. **Shuffled**: The passage list is fully shuffled before each self-consistency LLM call, giving m calls with m random passage perturbations.

3.2.2 Sub-batched

Sub-batching methods select subsets of passages for each batch, providing less context than all-in-one batching but also asking the LLM to generate fewer scores. We consider the following context selection and permutation strategies:

1. **Initial Order:** To create B batches, the initial list is partitioned into B non-overlapping intervals while maintaining its order (e.g., $[p_1, \cdots, p_{30}] \rightarrow [p_1 \cdots, p_{10}]$, $[p_{11} \cdots, p_{20}]$,

[3]The LLM context window must be large enough to fit all candidate passages, otherwise sub-batching (c.f. Sec 3.2.2) is required.

Table 1: AUC^m, representing the AUC-PR at m self-consistency calls/passage, for PW relevance assessment methods at $m = 1$ (no self-consistency) and $m = 15$. Increasing m improves all PW methods, but the batched PW methods improve more, becoming the best methods at $m = 15$, likely due to their larger prompt contexts. The highest AUC-PR for each m and LLM is in bold.

			Legal Search			DL-19			Covid		
	Psg/ Batch	Psg Order	GPT-4o AUC^1/ AUC^{15}	Sonnet AUC^1/ AUC^{15}	Nova AUC^1/ AUC^{15}	GPT-4o AUC^1/ AUC^{15}	Sonnet AUC^1/ AUC^{15}	Nova AUC^1/ AUC^{15}	GPT-4o AUC^1/ AUC^{15}	Sonnet AUC^1/ AUC^{15}	Nova AUC^1/ AUC^{15}
Shallow (30 psg)	1	–	**32** / 41	**24** / 28	24 / 29	39 / 52	**27** / 31	29 / 33	**70** / 78	**68** / 73	**68** / 77
	10 (sub-batch)	Init.	31 / 44	22 / 27	**27** / 33	34 / 51	25 / 33	**32** / 40	70 / 77	63 / 70	68 / 74
		STB	31 / **46**	22 / **38**	25 / **39**	39 / 57	25 / **41**	30 / 47	69 / **80**	65 / **77**	68 / **79**
		BTS	31 / 45	23 / 35	25 / 36	39 / 55	25 / **41**	**32** / 46	69 / 79	63 / 75	65 / 78
	30 (all psgs)	Init.	**32** / 45	23 / 27	**27** / 37	**46** / 55	26 / 35	30 / 44	69 / 78	63 / 69	63 / 74
		Shuf.	30 / 45	20 / 32	24 / 38	37 / **58**	24 / **41**	31 / **53**	69 / **80**	58 / 74	65 / 77
Deep (90 psg)	1	–	20 / 29	**15** / 20	15 / 20	**34** / 51	**23** / 28	25 / 31	63 / 72	**62** / 68	**62** / 70
	30 (sub-batch)	Init.	20 / 34	13 / 21	**18** / 28	30 / 49	20 / 30	**29** / 40	**64** / 75	55 / 67	60 / 72
		STB	20 / **36**	14 / **27**	17 / **31**	32 / **52**	19 / 40	23 / 42	**64** / **77**	56 / **73**	60 / **75**
		BTS	**22** / **36**	14 / 26	17 / 29	29 / **52**	18 / 37	25 / **44**	**64** / **77**	55 / 72	60 / **75**
	90 (all psg)	Init.	16 / 26	11 / 16	13 / 20	22 / 48	16 / 33	23 / 37	52 / 67	48 / 57	45 / 54
		Shuf.	14 / 29	11 / 21	13 / 20	22 / 50	15 / **42**	19 / **44**	53 / 73	45 / 65	46 / 60

$[p_{21}, \cdots, p_{30}]$, where p_i is the i'th passage in the initial list). As shown in the top right of Figure 1, self-consistency calls for each sub-batch are identical, giving this sub-batching strategy the least diversity in LLM sampling.

2. **Shuffled-then-Batched (STB):** The passage list is fully shuffled and then split into B batches before each LLM call. When self-consistency is used, STB creates the most diverse range of passage subsets and permutations across LLM calls.

3. **Batched-then-Shuffled (BTS):** The initial list is first divided into B intervals, as in Initial Order Sub-batching above, and then each batch b is fully shuffled before each LLM call. While BTS attempts to mitigate position bias through shuflling, the passage mixture for a given batch remains constant across self-consistency calls, making its LLM sampling less diverse than STB.

4 Experimental Setup

We evaluate the effects of batching and self-consistency on the relevance assessment and ranking abilities of three LLMs, namely GPT-4o (128K), Sonnet 3 (200K), and Amazon Nova Pro (300K) at temperature 1 across three passage retrieval datasets, releasing anonymized code.[4]

Self-consistency: The tested number of self-consistency calls per passage, m, ranges from 1 (i.e., no self-consistency) to 15. We ensure that for each query, each passage appears in exactly m calls for all LLM methods.

4.1 Passage Relevance Assessment and Ranking Tasks

Each passage retrieval task contains a set of queries $\mathcal{Q}$, a corpus of passages $\mathcal{D}$, and a relevance label $y_{q,p} \in \mathbb{R}$ for each query-passage pair. For each q, we also have an list of D passages $L^q = [p_1, \cdots, p_D]$ returned by some initial retrieval algorithm (e.g., BM25, dense retrieval).

Metrics: We use NDCG@10 to evaluate LLM ranking quality. We treat relevance assessment (PW methods only) as a binary classification task, first converting each LLM score $s_{q,p} \in [0, 3]$ to a relevance probability as $p(\hat{y}_{q,p} = 1) = s_{q,p}/3$, and then using the area under the precision recall curve (AUC-PR) for evaluation.[5]

Shallow vs. Deep Search To study the effect of the total number of passages evaluated, we test a short ($D = 30$) and long ($D = 90$) initial list length in what we call *shallow* and *deep* search, respectively. All sub-batching methods split the initial list into three equal sized batches (i.e., 10 and 30 passages per batch for shallow and deep search, respectively). The longest resulting batch (90 TREC Covid passages) is roughly 24K words – well within the context limits of all three LLMs.

[4] https://anonymous.4open.science/r/batched-sc-emnlp/

[5] AUC-PR is preferred to AUC-ROC for imbalanced data (Saito and Rehmsmeier, 2015), and IR datasets are highly imbalanced.

4.2 Datasets

Our evaluation includes two well-known open-source datasets, TREC DL-19[6] and TREC Covid (Thakur et al., 2021) with BM25 used to retrieve the initial passage list L^q for each q. We also test a third, closed-source dataset from a large legal technology company which we call Legal Search. Legal Search comprises 100 legal queries and passages chunked from 100,000 proprietary legal documents and uses the output of a multi-stage industrial retrieval pipeline to retrieve L^q. Using both open- and closed-source data allows us to examine whether our findings generalize between data that will have been seen by the LLMs during pretraining and data which has not (Törnberg, 2024).

5 Experimental Results

5.1 RQ1: Effects of Batching and Self-Consistency on Relevance Assessment

Figures 2 and 3 and Table 1 show the effects of increasing the number of self-consistency LLM calls/passage (m) from 1 (i.e. no self-consistency) to 15 on relevance assessment performance (AUC-PR), with more plots shown in Appendix A. While one-by-one PW methods are competitive at $m = 1$, they are always outperformed by a batched PW method at $m = 15$.

Batching can amplify the benefits of self-consistency: Though increasing the number of self-consistency calls greatly improved all PW methods, batched methods improved considerably more than one-by-one methods. We conjecture this effect may be due to the larger context created by batching leading to more diverse LLM sampling across multiple self-consistency calls. For instance, for GPT-4o on Legal Search (Shallow), one-by-one PW improved from 32% AUC-PR at $m = 1$ to 41% at $m = 15$ ($+9\%$), while all-in-one PW (Shuffled) improved from 30% to 45% ($+15\%$), respectively. Remarkably, for our shallow setting ($D = 30$), all-in-one PW uses 30 times fewer LLM calls per query than one-by-one PW.

Shuffling is helpful for high-enough m: At $m = 15$, the best performance acoss all datasets and LLMs was always achieved by a batched method with shuffling, which can likely be attributed to more diverse LLM sampling with reduced position bias across self-consistency calls.

[6] https://github.com/microsoft/TREC-2019-Deep-Learning

Sub-batching is useful for deep search: While all batched self-consistency methods performed well in shallow search (30 passages), for deep search (90 passages), the all-in-one methods performed poorly – far worse than the sub-batched methods (green lines in Figure 3). RQ2 further explores why large batches of 90 passages degraded performance at the values of m tested.

5.2 RQ2: Effects of Position Biases

Before considering the effect of the initial passage order (RQ3), we first ask whether batched LLM scoring exhibits consistent positional biases (e.g., the first few passages in a batch are scored higher) across random permutations of a given passage list. Addressing this question, Figure 4 shows LLM scores versus passage positions in a batch, with each passage seen by an LLM in $m = 15$ random permutations (90 total passages).

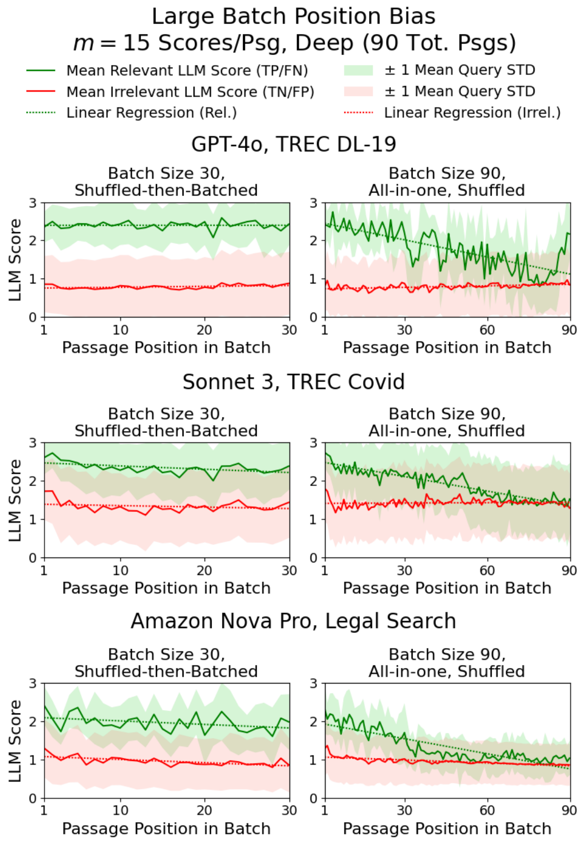

Figure 4: Mean relevant and irrelevant LLM scores with sub-batches of 30 psgs/batch (left) vs. all-in-one batches of 90 psgs/batch (right). The sub-batches are relatively consistent in discriminating relevance throughout the batch, while the all-in-one batches lose most of their discriminative power towards the tail of the batch.

Large batches have harmful position biases that can be mitigated by sub-batching: The harmful biases in the large 90 passage batches in Figure 4 are obvious: the capacity to discriminate between relevant vs. non-relevant passages is almost gone in the area towards the tail of the batch, with GPT-4o showing a clear lost-in-the-middle effect (Liu et al., 2024). By comparison, sub-batching with 30 passages per batch is far more consistent in being able to discriminate relevance throughout the batch, explaining its far superior performance on deep search.

5.3 RQ3: Effects of Initial Passage Order

Next we investigate the potential positional biases caused by the initial list order L^q. Figure 5 compares several batched methods that use the initial order to one-by-one LLM scoring, which does not depend on the order of L^q.

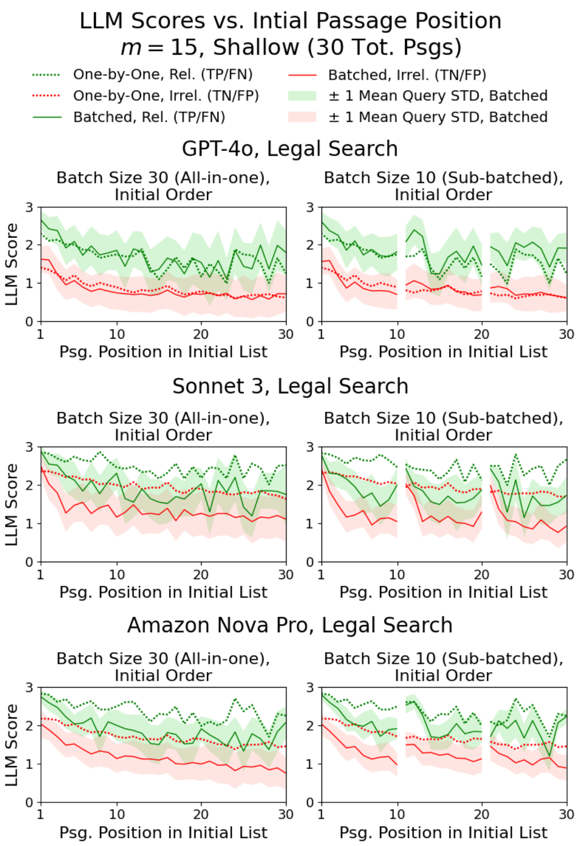

Figure 5: Mean LLM scores for one-by-one PW versus batched PW methods with initial order. For all-in-one PW methods, GPT-4o tracks much more closely to one-by-one PW than Sonnet and Nova Pro. Sub-batching with initial order creates artificial cycles and discontinuities at batch junctions.

GPT-4o has the least batching bias: For GPT 4o all-in-one PW in Figure 5, though the front and tail of the batch have slightly higher scores, overall, the batched scores track quite closely to one-by-one PW scores. In contrast, batched Sonnet and Nova Pro scores are typically far lower than their respective one-by-one scores everywhere except at the very front of the batch, helping explain the weakness of these LLMs.

Sub-batching with initial order can induce cycles and discontinuities The RHS of Figure 5 shows that sub-batching with initial order can cause score peaks at the start of every sub-batch, creating discontinuities at batch junctions, and inducing cyclical score fluctuations. This explains why sub-batching with the initial order performs worse than the shuffling variants.

5.4 RQ4: Ranking Performance of Batched Self Consistency Methods

Ranking performance in terms of NDCG@10 for all LLM methods is shown in Tables 2 and 3 for shallow and deep search, respectively, with detailed results on the effects of m in Appendix B.

Batching amplifies self-consistency benefits for ranking: Without self consistency ($m = 1$), one-by-one PW and LW (initial order) methods are competitive rankers, but adding self-consistency helps batched PW methods more than it helps these baselines, making the batched PW methods with $m = 15$ the strongest rankers overall. For instance, as seen in Table 3 for Legal Search (deep), GPT-4o one-by-one PW ranking improves from 44.9% NDCG@10 with $m = 1$ to 46.8% with $m = 15$, while sub-batched (STB) PW ranking improves from 43.8% to 51.3%, respectively (while needing 30 times fewer LLM calls).

STB ($m = 15$) performs best: Sub-batched STB methods with $m = 15$ performed best overall, likely due to creating the broadest range of contexts for LLM sampling by creating the most diverse passage permutations and subsets. All-in-one batched PW methods (which needed 3 times fewer LLM calls than STB) with $m = 15$ were also effective for shallow search (30 passages), but under-performed for deep search (90 passages), likely due to the harmful large-batch biases seen in Figure 4.

Table 2: NDCG@10 (%) for shallow reranking (30 total passages) for all LW and PW LLM methods with $m = 1$ (i.e., no self-consistency) $m = 15$ (i.e., 15 self-consistency calls/psg). LW and one-by-one PW methods are competitive at $m = 1$ but do not benefit as strongly from self-consistency as batched PW methods, causing the later to achieve the best NDCG@10 at $m = 15$.

	Psgs/		Psg	Legal Search			DL-19			Covid		
Method	**Batch**	m	**Order**	**GPT-4o**	**Sonnet**	**Nova**	**GPT-4o**	**Sonnet**	**Nova**	**GPT-4o**	**Sonnet**	**Nova**
Initial	–	–	–	37.2	37.2	37.2	50.6	50.6	50.6	59.5	59.5	59.5
LW	30	1	Init.	46.2	41.6	44.5	65.9	65.2	61.9	73.4	66.4	67.9
			Shuf.	13.9	12.6	13.8	34.6	36.0	34.6	45.7	47.5	48.3
		15	Init.	48.9	45.1	46.7	67.4	66.7	63.6	74.6	70.5	70.3
			Shuf.	13.9	15.3	13.3	33.0	30.4	30.8	45.0	47.4	45.2
PW	1	1	–	45.6	42.7	41.7	63.3	63.4	64.1	75.2	73.9	75.7
		15	–	46.5	44.5	43.0	67.8	65.4	64.7	76.6	76.0	78.8
	10	1	Init.	45.5	40.2	42.4	65.6	63.5	65.9	75.6	72.7	75.8
			STB	45.5	38.5	42.8	67.5	63.0	64.8	74.7	74.6	75.0
			BTS	45.3	39.2	42.1	66.6	63.0	67.0	75.4	73.3	75.5
		15	Init.	48.5	38.9	43.6	67.8	64.9	67.4	77.9	74.2	76.6
			STB	**50.0**	**45.4**	48.2	68.6	**67.8**	68.0	**80.7**	**79.6**	**80.4**
			BTS	48.1	43.1	45.4	68.9	67.0	**68.8**	78.7	79.5	79.6
	30	1	Init.	46.1	42.8	43.6	66.5	63.4	65.6	76.1	73.2	73.6
			Shuf.	44.5	34.8	43.9	66.7	64.1	64.3	75.6	69.5	73.3
		15	Init.	49.4	41.8	46.9	**69.3**	65.6	68.0	77.9	74.3	76.7
			Shuf.	**50.0**	41.3	**48.3**	68.6	67.7	67.6	80.3	77.2	79.1

Table 3: NDCG@10 (%) for deep reranking (90 total passages) for all LW and PW methods at $m \in \{1, 15\}$. Sub-batched methods (30 psg/batch) perform best at $m = 15$ with the STB variant typically achieving the highest NDCG@10, likely due having the most diverse batching strategy and avoiding large-batch position biases.

	Psgs/		Psg	Legal Search			DL-19			Covid		
Method	**Batch**	m	**Order**	**GPT-4o**	**Sonnet**	**Nova**	**GPT-4o**	**Sonnet**	**Nova**	**GPT-4o**	**Sonnet**	**Nova**
Initial	–	–	–	37.2	37.2	37.2	50.6	50.6	50.6	59.5	59.5	59.5
LW	90	1	Init.	46.0	41.1	36.4	70.3	64.6	56.1	76.2	67.1	64.1
			Shuf.	9.4	8.6	9.4	23.5	25.5	23.8	42.0	39.4	39.5
		15	Init.	48.4	42.5	42.4	72.5	66.7	67.2	65.2	60.1	63.3
			Shuf.	13.2	12.7	12.8	26.3	25.4	25.7	44.5	45.2	44.3
PW	1	1	–	44.9	41.8	41.7	69.8	66.4	67.7	78.9	77.8	80.0
		15	–	46.8	**43.5**	42.9	73.6	68.6	69.3	80.1	79.8	83.3
	30	1	Init.	43.6	38.2	42.8	72.3	62.0	68.7	82.9	72.2	75.6
			STB	43.8	34.9	44.7	70.7	63.2	68.7	79.8	72.6	78.2
			BTS	40.4	33.7	42.7	69.5	64.0	67.6	81.6	71.6	75.2
		15	Init.	47.1	36.3	44.1	71.0	66.0	69.5	83.8	77.4	79.9
			STB	**51.3**	41.3	**49.8**	**76.3**	**71.9**	**72.1**	86.1	82.1	**84.3**
			BTS	50.6	39.2	46.7	73.9	70.0	70.2	**86.3**	**82.5**	83.5
	90	1	Init.	43.5	37.7	41.9	66.9	62.2	64.4	73.8	71.8	68.6
			Shuf.	29.0	26.4	32.7	55.4	52.5	51.3	63.3	61.3	59.0
		15	Init.	48.5	39.7	45.0	72.1	66.7	66.3	79.3	77.0	72.5
			Shuf.	45.1	28.4	40.4	73.0	61.2	65.9	82.6	71.8	74.4

LW ranking with the initial list order is competitive with one-by-one PW ranking: When the initial list order L^q is kept, LW ranking is competitive with one-by-one PW ranking, but when L^q is shuffled, LW methods perform very poorly. Upon examining the LLM outputs, we found that for multiple shuffled input lists $[p_1, p_2, ...]$ where p_i is the i'th passage in the batch (i.e., a random passage), the LLM often generated the same rankings (e.g., $[p_3, p_{12}, p_{13}, ...]$), thus using only the input passage position and not the content. This effect did not occur when initial order was kept.

6 Conclusion

We propose that batched PW LLM relevance assessment and ranking, where multiple candidates are judged in one LLM call, is especially well-suited for self-consistency. Through multiple experiments, we show that batching amplifies self-consistency benefits, leading to SOTA performance. We conjecture this is because large, diverse input contexts (enabled by candidate batching) increase the diversity of LLM sampling over multiple self-consistency calls.

Limitations

The main limitation of our work are the computational resources required, since LLM relevance assessment and ranking is expensive computationally. We thus only tested a range of $m \in \{1, \cdots, 15\}$ for the number of self-consistency calls, even though higher levels of m would have added information to our results. Also due to computational limitations, we only tested three LLMs (GPT-4o, Claude Sonnet 3, and Amazon Nova Pro) across three datasets (TREC DL-19, TREC Covid (Thakur et al., 2021), and Legal Search), though it would be interesting to test an even wider range of models and datasets. Simlarly, we were limited to testing four batch sizes $\in \{1, 10, 30, 90\}$ across five batching strategies (c.f. Figure 1) and two levels of search: shallow (30 initial passages) and deep (90 initial passages).

As another limitation, we note that LLMs likely will have seen open-source datasets such as TREC DL-19 and TREC Covid during pretraining, which is why using the third, closed-source Legal Search dataset is very important in our experiments. Fortunately, we are able to observe that our results generalize across both the open-source and closed-source data.

Finally, we must point out several risks of using LLMs for ranking and relevance assessment at scale. Firstly, LLMs can amplify societal biases that they will have learned during their pretraining process, creating a risk for harm. Secondly, LLMs carry a risk of "jail-breaking", or malicious prompt injection, creating safety risks. Finally, LLMs may provide incorrect judgments on passage relevance, which could have severely negative effects for high-stakes applications.

References

Luyu Gao and Jamie Callan. 2021. Condenser: a pre-training architecture for dense retrieval. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 981–993.

Yupeng Hou, Junjie Zhang, Zihan Lin, Hongyu Lu, Ruobing Xie, Julian McAuley, and Wayne Xin Zhao. 2024. Large language models are zero-shot rankers for recommender systems. In *European Conference on Information Retrieval*, pages 364–381. Springer.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2021. Unsupervised dense information retrieval with contrastive learning. *arXiv preprint arXiv:2112.09118*.

Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. 2024. More agents is all you need. *arXiv preprint arXiv:2402.05120*.

Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Adian Liusie, Potsawee Manakul, and Mark Gales. 2024. Llm comparative assessment: Zero-shot nlg evaluation through pairwise comparisons using large language models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 139–151.

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 *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8086–8098.

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

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *arXiv preprint arXiv:1901.04085*.

Zhen Qin, Rolf Jagerman, Kai Hui, Honglei Zhuang, Junru Wu, Le Yan, Jiaming Shen, Tianqi Liu, Jialu Liu, Donald Metzler, and 1 others. 2024. Large language models are effective text rankers with pairwise ranking prompting. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 1504–1518.

Stephen Robertson, Hugo Zaragoza, and 1 others. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Takaya Saito and Marc Rehmsmeier. 2015. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432.

Gerard Salton, Anita Wong, and Chung-Shu Yang. 1975. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620.

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 *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 14918–14937.

Raphael Tang, Crystina Zhang, Xueguang Ma, Jimmy Lin, and Ferhan Türe. 2024. Found in the middle: Permutation self-consistency improves listwise ranking in large language models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 2327–2340.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *arXiv preprint arXiv:2104.08663*.

Paul Thomas, Seth Spielman, Nick Craswell, and Bhaskar Mitra. 2024. Large language models can accurately predict searcher preferences. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1930–1940.

Petter Törnberg. 2024. Best practices for text annotation with large language models. *arXiv preprint arXiv:2402.05129*.

Shivani Upadhyay, Ronak Pradeep, Nandan Thakur, Nick Craswell, and Jimmy Lin. 2024. Umbrela: Umbrela is the (open-source reproduction of the) bing relevance assessor. *arXiv preprint arXiv:2406.06519*.

Fanqi Wan, Xinting Huang, Deng Cai, Xiaojun Quan, Wei Bi, and Shuming Shi. 2024. Knowledge fusion of large language models. *arXiv preprint arXiv:2401.10491*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, and 1 others. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

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 *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2308–2313.

A Appendix A: Relevance Assessment Quality vs Scores per Passage

Figures 6 and 7 bellow show the effects of m on AUC-PR of one-by-one PW and batched PW methods for all datasets and LLMs for shallow and deep search, respectively.

B Appendix B: NDCG@10 vs Scores per Passage

Figures 8 and 9 bellow show the effects of m on NDCG@10 of one-by-one PW and batched PW methods for all datasets and LLMs for shallow and deep search, respectively.

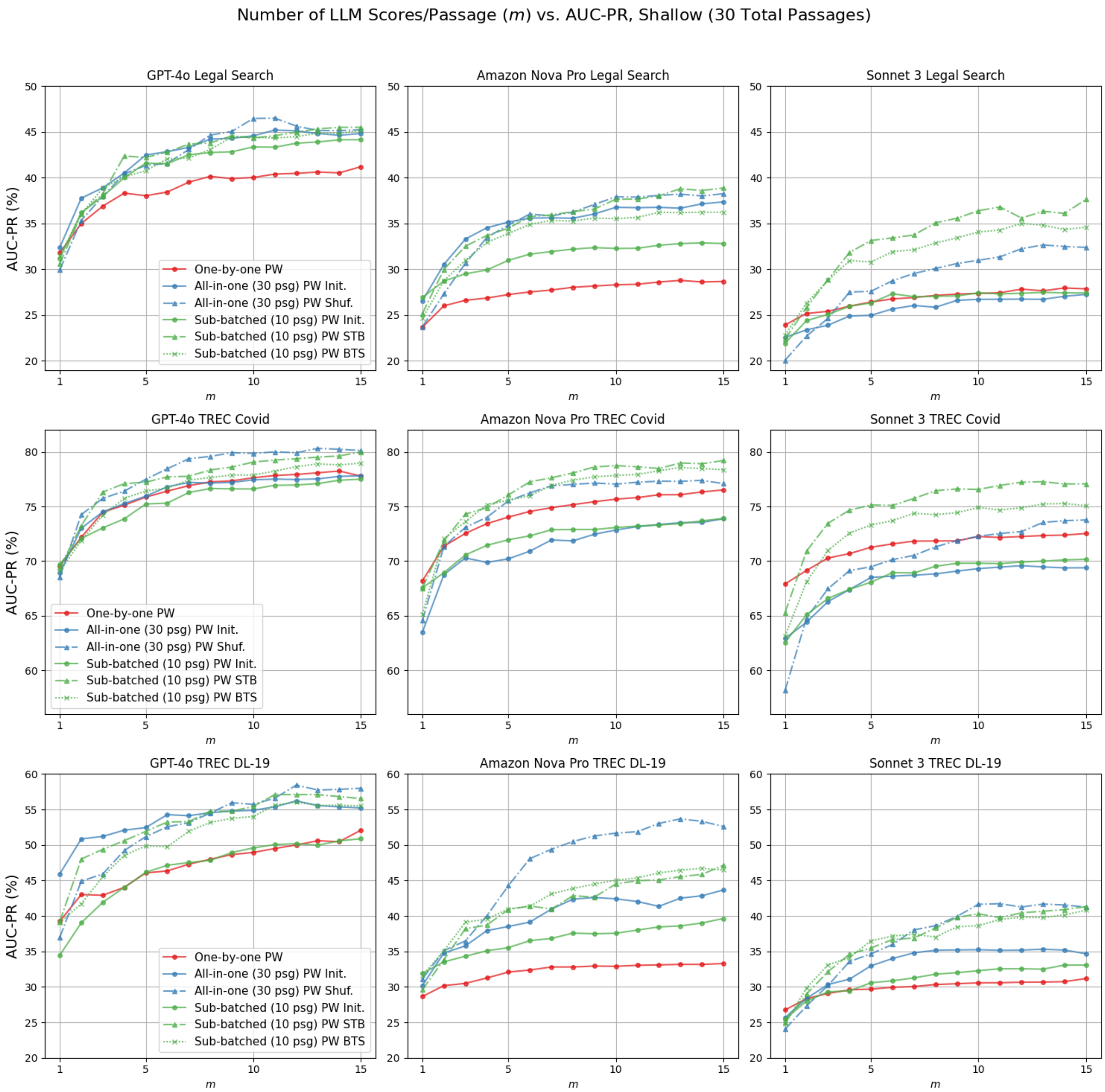

Figure 6: Number of LLM Scores/Passage (m) vs. AUC-PR, Shallow (30 Total Passages)

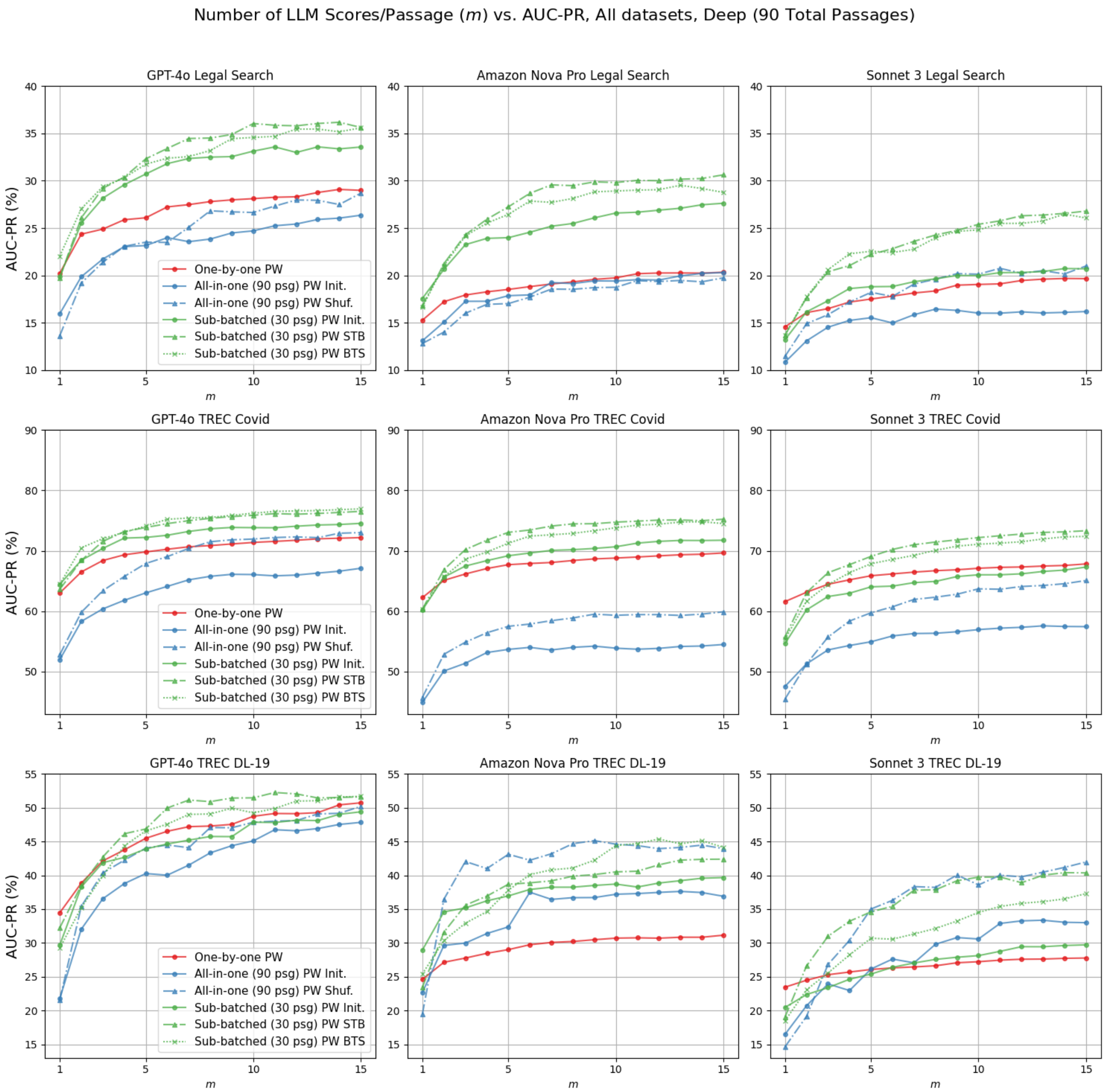

Figure 7: Number of LLM Scores/Passage (m) vs. AUC-PR, Deep (90 Total Passages)

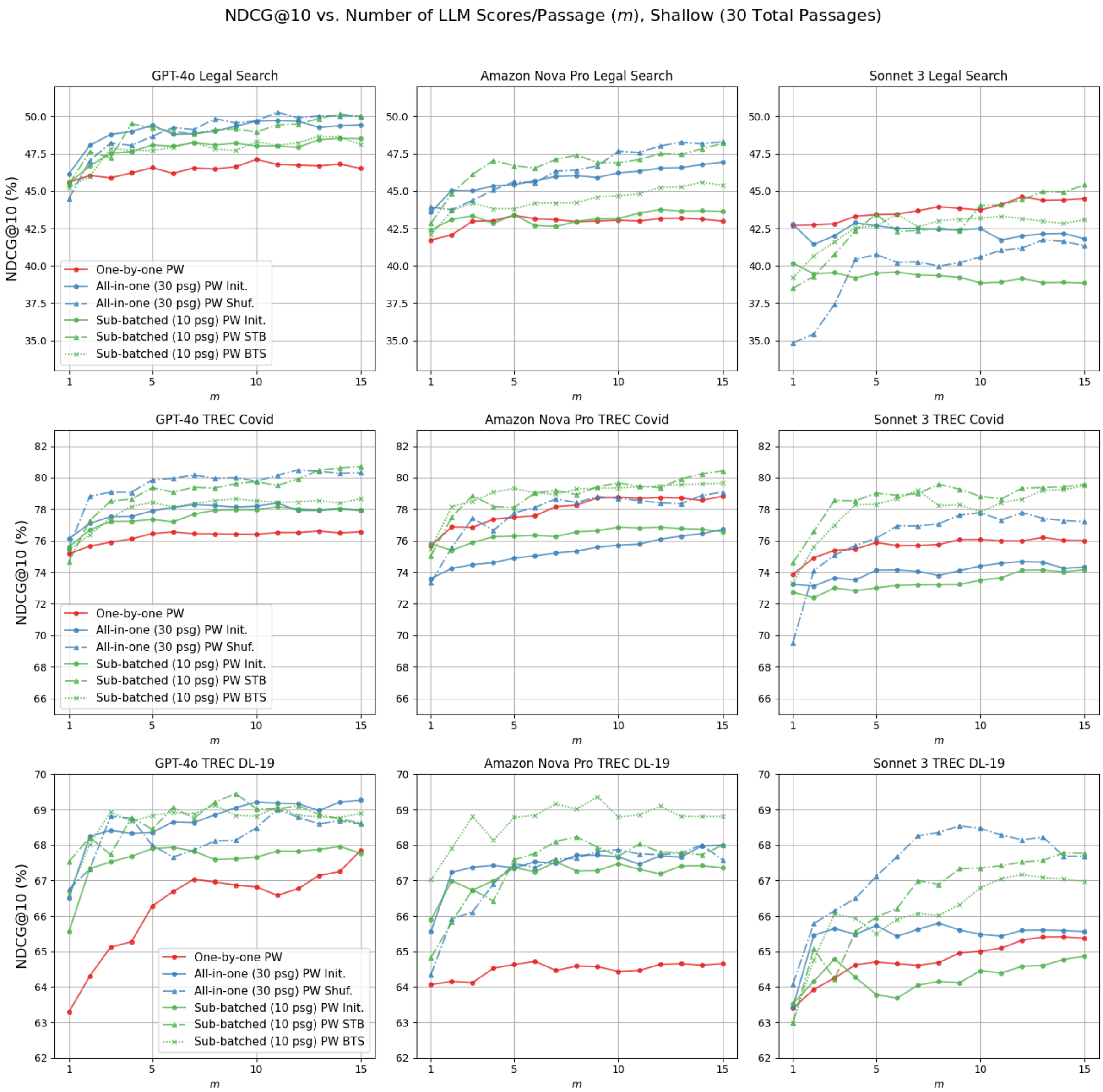

Figure 8: Number of LLM Scores/Passage (m) vs. NDCG@10, Shallow (30 Total Passages)

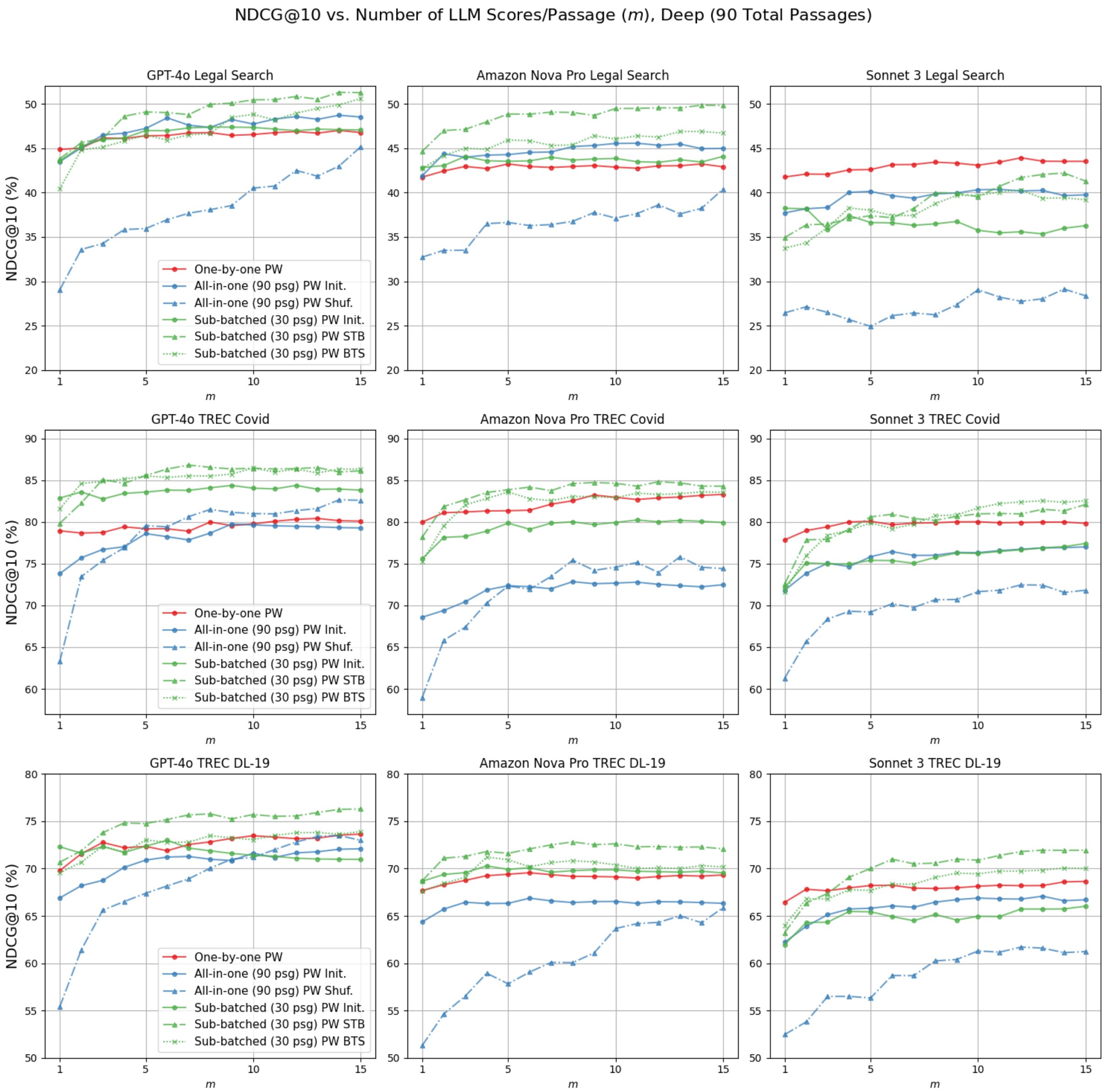

Figure 9: Number of LLM Scores/Passage (m) vs.NDCG@10, Deep (90 Total Passages)

C Appendix C: Prompt Templates

Figures 10 and 11 show the full prompts used for our PW and LW implementations.

```
Assign one of the labels (i.e., integer scores) below to
each of the {{list_len}} passages based on its relevance
to the query. Following the order of the passages below,
output your answer as only a list of {{list_len}} labels.

<Label Instructions>
3 - The passage is dedicated to the query and contains
the exact answer.
2 - The passage has some answer for the query, but the
answer may be a bit unclear, or hidden amongst extraneous
information.
1 - The passage seems related to the query but does not
answer it.
0 - The passage has nothing to do with the query.
</Label Instructions>

<Query>
{{query}}
</Query>

<Passages>
{% for p_id, p_text in passages %}
  {{ p_id }}: {{ p_text }}
{% endfor %}
</Passages>

<Example Output Format>
[<score for p1>, <score for p2>, ...]
</Example Output Format>

<Output>
```

Figure 10: The pointwise relevance assessment prompt, based on the relevance label instructions from the UMBRELLA open source reproduction of the Bing relevance assessment prompt (Upadhyay et al., 2024).

```
Return the best ordering of the passages given the query.
Output only a list of passage ID strings with a list
length of exactly k ids.

<Example>
<Example Query>
<example query text>
</Example Query>

<Example Passages>
  p1: <1st passage>
  p2: <2nd passage>
  p3: <3rd passage>
</Example Passages>

<Example k>
3
</Example k>

<Example Output List>
Output list: [<most relevant pid>,<2nd-most relevant
pid>,<3rd-most relevant pid>]
</Example Output List>
</Example>

<Query>
{{query}}
</Query>

<Passages>
{% for p_id, p_text in passages %}
  {{ p_id }}: {{ p_text }}
{% endfor %}
</Passages>

<k>
{{k}}
</k>

<Output List>
Output list:
```

Figure 11: The listwise ranking prompt.