CORPUSWALKER: REASONING-GUIDED GRAPH EXPLORATION FOR SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Retrieve-then-rerank pipelines are limited by the recall of the first-stage retriever. Graph-based adaptive reranking alleviates this issue by expanding the candidate pool at test time. Existing adaptive reranking approaches are driven by the reranker's relevance score as a feedback signal over a single, fixed neighborhood graph. This poses a challenge for reasoning-intensive retrieval tasks (such as those covered in the BRIGHT benchmark) because reasoning-intensive relevance depends on the reasoning path needed to arrive at the answer. This breaks the assumption of standard reranker-guided graph exploration techniques, since similar documents are not necessarily both relevant in a reasoning setting. In this work, we argue that graph exploration is itself a reasoning task. Therefore, we propose CORPUSWALKER, which separates graph exploration from document ranking. During traversal, an LLM agent reads offline-extracted document features of the current pool and, at each turn, reasons about information needs, chooses which document to expand and the corresponding relation to follow. The trace of this exploration tracks the documents the agent expanded, and the corresponding reasoning serves as context for our Meta Reranker, which ranks the entire pool in one pass. On reasoning-intensive retrieval benchmarks (BRIGHT and FreshStack), CORPUSWALKER outperforms retrieve-then-rerank and graph-based adaptive reranking baselines under the same document budget, improving nDCG@10 on BRIGHT by 3.1 points and Recall@100 by 4.7 points over the strongest of them, with up to 2.8 points gain in nugget coverage on FreshStack.

1 INTRODUCTION

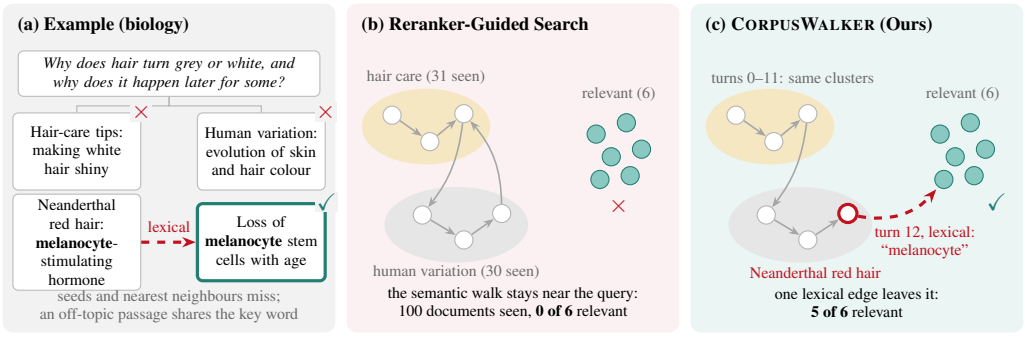


Figure 1: Example of a BRIGHT biology query. (a) The relevant passages explain hair greying by the loss of *melanocyte* stem cells; (b) Reranker-Guided Search walks the embeddings graph from the seeds and stays within these two clusters, and none is relevant. (c) CORPUSWALKER explores the same clusters for a few turns, then expands an off-topic passage using *lexical* edge.

Many retrieval tasks require an understanding and reasoning over background information to determine a document's relevance to a query. These *reasoning-intensive retrieval* tasks, such as those measured by the BRIGHT (SU et al., 2025) and FreshStack (Thakur et al., 2025) benchmarks, have attracted substantial attention because of their difficulty. Methods to address these tasks primarily focus on fine-tuning models that can perform the reasoning required to make difficult relevance

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

estimates (e.g., Weller et al. (2025b); Shao et al. (2025)). These models are then leveraged as either retrievers or rerankers in "retrieve-then-rerank" pipelines, which use the retriever to filter down a collection to a relatively small number of documents that can be evaluated by a more capable reranker.

Retrieve-then-rerank pipelines are ultimately limited by the recall of the first-stage retriever, since the retriever acts as a filter, and documents are removed by this filter are not considered in the final ranking. This is especially problematic for reasoning-intensive retrieval tasks, since the model may not have the required background before performing the filtering. Graph-based adaptive reranking methods, which were originally proposed for semantic retrieval (MacAvaney et al., 2022) and more recently applied to reasoning-intensive retrieval (Xu & Chen, 2026; Rathee et al., 2026), address the bounded recall problem of retrieve-then-rerank pipelines by using feedback signals from the reranker to add similar (and potentially-relevant) documents that were filtered out by the first stage by navigating a nearest-neighbor graph of the collection. However, using reranker scores as a feedback signal is suboptimal in reasoning-intensive retrieval tasks is not ideal because the nearest neighbors of relevant documents are often themselves not relevant. We show a concrete example of a BRIGHT Biology query in Figure 1.

In this paper, we argue that for reasoning-intensive retrieval, graph exploration itself benefits from reasoning capabilities. To test this hypothesis, we propose CORPUSWALKER, a new LLM-based agentic method for exploring a collection's nearest neighbor graph in adaptive reranking pipelines. Unlike prior adaptive methods, CORPUSWALKER operates separately from the reranker, instead determining how to expand the re-ranking pool before the reranker independently determines the final ranking of the expanded document set. To allow further autonomy in the exploration process, CORPUSWALKER can be given access to multiple nearest neighbor graphs (we explore both lexical and semantic graphs), allowing it to determine which types of relationships to follow when. To manage online computational cost, the system also operates over summaries of the documents (computed offline), rather than their original text. Extensive experiments across both BRIGHT and FreshStack show that the method consistently improves reasoning-intensive retrieval with up to 3.1 points gain in ranking, 4.7 points gain in retrieval performance (on BRIGHT), and 2.7 points gain in nugget coverage (on FreshStack). Appendix A shows an example trace of CORPUSWALKER for a query where the system overcame low recall and improved nDCG@10 from 0.0 to 83.8.

In summary, we introduce a new perspective on how to leverage reasoning for retrieval, by using it to explore the corpus instead of only making relevance estimations on individual documents. We test this hypothesis through CORPUSWALKER, which introduces two key novelties: (i) leveraging an agent to explore a cheap multi-relational corpus graph to choose both which documents and which types of relations to follow, and (ii) separating exploration from ranking by using a separate MetaRanker, which operates as a single LLM call and reuses the exploration trace as evidence. Extensive experiments show its strong, consistent value for reasoning-intensive retrieval. Our code is available at `https://anonymous.4open.science/r/CorpusWalker-7CCD/`.

2 RELATED WORK

2.1 REASONING-BASED RETRIEVAL AND RANKING

Going beyond standard information-seeking tasks, more recently much interest and effort have been garnered around reasoning-intensive retrieval tasks (SU et al., 2025; Thakur et al., 2025; Weller et al., 2025a). These retrieval tasks require more than surface form matching or semantic matching, unlike standard information-seeking tasks. Such reasoning-intensive retrieval tasks require deliberate reasoning for discerning the underlying logic that is required to answer the question. For instance, proving a theorem requires retrieving documents that may specify a related or core underlying logic that helps guide the proof mechanism. Similarly, a coding task may require the ranker to match based on the logic required to solve problem, rather than just keyword and semantic matching.

LLMs (Large Language Models) have been adopted as retrieval and re-rankers (Li et al., 2025a; Gangi Reddy et al., 2024; Ma et al., 2024; Weller et al., 2025a; Asai et al., 2023; Muennighoff et al., 2025; Shao et al., 2025) which go beyond surface form or phrase-level matching. Reasoning LLM-based rerankers such as Rank1 (Weller et al., 2025b), REARank Zhang et al. (2025a), and Rank-R1 (Zhuang et al., 2026) apply inference-time compute in IR by allotting more compute for

enhancing document re-ranking. Thinking-free ranker TFRank (Fan et al., 2026) further enhances relevance by avoiding thinking traces and generating relevance labels. Another line of work keeps the retriever and ranker fixed and performs reasoning over the queries. TongSearch-QR (Qin et al., 2025) learns a reasoning-style rewrite with reinforcement learning, and ReDI (Zhong et al., 2025) decomposes the query into interpreted sub-queries. On the other hand, ThinkQE (Lei et al., 2025) and DIVER-QExpand (Long et al., 2025) used the retrieved documents iteratively to refine the query representation, as in pseudo-relevance feedback.

Following the agentic capabilities of LLMs, the standard search has evolved to take autonomous search actions (Li et al., 2025b; Song et al., 2025; Jin et al., 2025), e.g., query planning, retrieval when needed, or iterative loops. Further, the use of tools has led to direct interaction with the corpus Li et al. (2026) during search (DCI), which reads the raw text from the corpus instead of the index using general-purpose terminal tools like grep. However, DCI incurs high latency per query (often exceeding an hour per query), since it involves expensive grep-style matching over a potentially large raw corpus multiple times per query.

2.2 Graph-based Search

Classical retrieve-rerank pipelines usually adopt a cascading setting where first-stage retrieval (Robertson & Zaragoza, 2009; Formal et al., 2021; Wang et al., 2011) provides fast but imprecise relevance estimates. This is followed by a heuristic top-k filtering of documents based on retrieval scores which are then provided to a reranker. However, the traditional approaches suffer from the "bounded recall" problem where the re-ranker is ultimately limited by the recall of the first-stage retriever (MacAvaney et al., 2022; Rathee et al., 2025).

To address this gap, graph-based search approaches such as GAR (MacAvaney et al., 2022), SlideGAR (Rathee et al., 2025), RGS (Xu & Chen, 2026) and REPAIR (Kim et al., 2026) have been proposed that construct a document-document corpus graph offline. To construct such corpus graphs, a document is treated as a query and top results are neighbours. Such corpus graphs are very efficient and easy to scale to millions of documents, as shown in GAR and SlideGAR. During reranking, these approaches traverse this graph at inference time, expanding to new documents beyond the initial retrieval pool with the help of feedback signal from the re-ranker. RGS (Xu & Chen, 2026) also employs a similar heuristic greedy search over the corpus graph. However, this traversal is purely guided by the ranker's relevance. Recent methods like GAREN (Park et al., 2026) and LATTICE (Gupta et al., 2026) proposed building more fine-grained graphs, such as group partitioning or hierarchical index trees, from LLM-generated multi-level document summaries. Our method uses an existing kNN/lexical graph instead of an LLM-built tree, and chooses one expansion per turn with explicit reasoning, under a fixed rerank budget.

Graph-based search methods help tackle recall boundedness issues, they still operate with a **single fixed static graph**, rely on heuristic exploration of the graph, graph creation strategy, and are constrained by semantic matching. As detailed in Section 2.1 these design choices are quite limiting for reasoning-intensive retrieval tasks that require going beyond surface form or semantic matching. However, our proposed CorpusWalker helps bridge these gaps by going beyond semantic matching. It achieves this by casting graph traversal as a reasoning task in a graph-based search setup, through a traversal agent. The agent adaptively determines which graph (lexical/semantic) to explore and also reasons to determine which edge to follow to add additional documents to the candidate pool through exploration. Unlike LATTICE and GAREN, our traversal agent explores the graphs under a fixed reranking budget and reads a fixed number of documents. Then a meta-ranker ranks the entire candidate pool in one pass conditioned on the additional context of the reasoning trajectory from the traversal agent.

3 Our Proposed Method: CorpusWalker

In this Section, we provide the core components of our method CorpusWalker namely: Information Compression (Section 3.1), Traversal Agent (Section 3.2), and Meta Reranker (Section 3.3). Finally, we give an overview of the main algorithm (Section 3.4) with an illustration in Figure 4.

3.1 INFORMATION COMPRESSION

In this section, we discuss our document summarization approach to improve the efficiency of ranking. Instead of ranking whole documents, we extract a compressed version of each document offline, incurring the cost of one LLM call per document. This incurs only a one-time cost for the whole corpus, as the compression is independent of the query, occurs offline, and is cached. We draw inspiration from the information-theoretic view of agentic systems discussed in He et al. (2026). It has also been adapted for document ranking (Tian et al., 2026). CORPUSWALKER adopts the compressor-predictor design pattern. This view primarily views information seeking as compressors that digest raw text and distill it into compact texts, which are consumed by a predictor which provides the final answer or, in our case, the final ranking. More formally, a compressor ρ takes the document d as input and produces a summary d_s ($d_s = \rho(.|d)$). The summary is consumed by a predictor, which provides an output $y = p(y|d_s)$. Mapping this to our setting, $\mathcal{A}$ and $\mathcal{R}$ from Algorithm 1 are the predictors in our setting, which consume the compressed version of the document to produce decisions regarding the next document to traverse and the final ranking, respectively. The prompt for the compressor is provided in Figure 10. The compression enables the whole candidate pool of documents obtained from the traversal agent to fit within the context of the ranker, requiring only one ranker call per query and also reducing input token costs (Section3.3). The fidelity of the ranker is bounded by the capability of ρ to capture useful information during compression. This is also in line with the observation made in (He et al., 2026), where the compressor quality (ability to retain task-relevant information) is more important than the parameter scale of the predictor.

3.2 TRAVERSAL AGENT

We use an LLM as the traversal agent, $\mathcal{A}$, which has access to the query and initial seed documents represented as compressed textual features. Since relevance needs reasoning, our agent first understands the underlying problem in the query and reads the seed document features. It compares the 2-3 strongest candidates and provides reasoning why a particular document can lead to the needed relevant documents. It also decides which edge-type expansion to take; mainly, it chooses from LEX and SEM. Since these two different graphs encode different properties, the agent utilizes this degree of freedom conditioned on the query needs at a particular turn. Some queries need exact keyword matching, such as an error message in a coding task or a theorem name in maths, and others might need semantic relevance. Since the traversal agent is not fine-tuned for this task, we give instructions in the prompt. We provide a detailed system prompt in Figure 8 in the Appendix. Also, the agent can be assigned a goal, for example, assembling a multi-part answer on FreshStack, and the agent takes decisions accordingly. We provide concrete examples along with traversal traces on a BRIGHT query in Figure 6 and on a FreshStack query in Figure 7 in the Appendix.

3.3 META RERANKER

The traversal agent outputs a pool, but the end goal is to provide document ranking. The default choice is a sliding-window listwise pass with window w and step b, it takes $1 + \lceil (c - w)/b \rceil$ LLM calls to cover a pool of c documents, and every call reads the pool from scratch. That pass throws away everything the traversal just learned. At the end of the traversal step, the agent has already read nearly the whole pool of c documents (except documents added in the last turn) as summaries, while reasoning about this specific query. However, the sliding window-based ranking would ignore what the walk has already established. Therefore, we replace the multi-window pass with a single call that is given the traversal reasoning walk as evidence. The prompt (Figure 9) contains the query, the agent's own reasoning traces, which consist of: for each turn, the document it expanded, the reason it gave, and how many new documents that expansion contributed, and all c pooled documents as cached summaries, each annotated with how it entered the pool: expanded by the agent at turn t, a seed, or discovered at turn t from expanding some parent that contributed n new documents. The Meta Reranker outputs only the document identifier permutation. Due to the size of offline extracted features, the ranking in one call is affordable. The whole pool is in the range of $8 - 10$k tokens at $c = 100$ and fits in a single prompt where raw document text would not. Ranking therefore costs one LLM call per query rather than multi-sliding-window LLM calls. In the compressor–predictor view, the ranker is a second predictor over the same compressed input the agent used: the representation's compactness is what makes one call possible, and its fidelity bounds what that call can recover.

Algorithm 1 CORPUSWALKER.

Require: query q; neighbour functions $\mathcal{N}_e(\cdot)$ for $e \in \{\text{LEX}, \text{SEM}\}$; rendering $\rho(\cdot)$; seed retriever R_0; seed size s; pool budget c; turn budget T; agent $\mathcal{A}$; ranker $\mathcal{R}$

1: $P \leftarrow R_0(q, s)$; $E \leftarrow \emptyset$; $\Gamma \leftarrow \langle\,\rangle$ ▷ pool, expanded set, trace
2: **for** $t = 1, \ldots, T$ **do**
3: **if** $|P| \geq c$ **or** $P \setminus E = \emptyset$ **then break**
4: **end if**
5: $(d^\star, e, r) \leftarrow \mathcal{A}\big(q,\ \{(d, \rho(d)) : d \in P \setminus E\},\ \Gamma\big)$ ▷ which document, which edge, and why
6: $E \leftarrow E \cup \{d^\star\}$
7: $P \leftarrow P \cup \textsc{Take}\big(\mathcal{N}_e(d^\star) \setminus P,\ c - |P|\big)$ ▷ select neighbours, to the budget
8: $\Gamma \leftarrow \Gamma \| \langle (t, d^\star, e, r) \rangle$
9: **end for**
10: **return** $\mathcal{R}\big(q,\ \Gamma,\ \{(d, \rho(d)) : d \in P\}\big)$ ▷ one call over the whole pool (§3.3)

3.4 ALGORITHM WALKTHROUGH

We present a detailed overview of Algorithm 1. CORPUSWALKER turns pool expansion into a sequence of decisions an LLM makes about the corpus rather than a rule applied to reranker scores. A first-stage retriever seeds a pool P with s documents (line 1), and the corpus is laid out as two graphs (LEX and SEM represent lexical and semantic, respectively) over the same documents, one encoding documents that share distinctive terms and one encoding documents close in embedding space. At each turn, the agent $\mathcal{A}$ is shown the query and unexpanded documents in the pool, each rendered by $\rho(\cdot)$ as short offline-extracted features together with the record of what it has already expanded and why. Traversal action returns three things (line 5): the document to expand, the relation to follow, and a reason why document was selected for expansion. The chosen document's neighbours along that relation are added to the pool (line 7) until the budget c is reached, and the turn is appended to the trace Γ (line 8). The loop stops at T turns, at the budget, or when every pooled document has been expanded. Finally, the Meta Reranker $\mathcal{R}$ ranks the whole pool conditioned on the traversal reasoning (line 10). The cost is at most $T + 1$ LLM calls per query and does not grow with the corpus, since $\rho(\cdot)$ is computed once offline and is query-independent.

4 EXPERIMENTAL SETUP

We perform extensive experiments to answer the following research questions:

RQ I: How effective is the reasoning-guided graph exploration in finding relevant documents?

RQ II: How does compaction /compression of evidence affect downstream ranking?

RQ III: How is the compute cost & robustness impacted with the traversal agent size?

Datasets. We use reasoning-intensive retrieval benchmarks like BRIGHT (SU et al., 2025) and FreshStack (Thakur et al., 2025) benchmarks for our evaluation. BRIGHT benchmark has 12 subsets that cover diverse domains such as general reasoning, maths, and coding, and measures the model's reasoning capabilities for complex search. FreshStack queries cover recent software libraries, and judges relevance per nugget of the answer. We provide more details in Tables 5 & 6 in the Appendix.

Retrieval and Rerankers. We use BM25 (Robertson & Zaragoza, 2009) and ReasonIR-8B (Shao et al., 2025) as initial retriever methods for our main experiments. We select a range of reranker from the pointwise and listwise families. We use TFRank-8B (Fan et al., 2026), Rank1-7B (Weller et al., 2025b), and Qwen3-8B-Reranker (Zhang et al., 2025b) as pointwise rankers and REARank-7B (Zhang et al., 2025a) and RankGPT (Sun et al., 2023) with Qwen3.6-35B-A3B Qwen Team (2026) as a listwise rankers.

Baselines Beyond retrieve-then-rerank, we compare our methods with graph expansion-based approaches, SlideGAR (Rathee et al., 2025) and RGS (Xu & Chen, 2026), which use initial retrieval seeds and explore the graph for finding relevant documents. These baselines mainly use the reranker feedback to explore the graph. However, beyond relevance, we use the traversal agent to expand the documents. All baseline methods use the same listwise reranker, Qwen3.6-35B-A3B. For BRIGHT,

we report nDCG@10 and Recall@10 for the final ranking quality and Recall@100 for evaluating pool quality. For FreshStack, we use the metrics introduced by its authors, α-nDCG@10 and Coverage@20, where α-nDCG rewards covering distinct answer nuggets and discounts repeated coverage of the same nugget, while Coverage is the fraction of a query's nuggets found. Further details are provided in Appendix D

5 RESULTS

5.1 SEARCH POOL QUALITY

To answer RQ I, we run our traversal agent on the BRIGHT benchmark and evaluate the pool quality as shown in Figure 2. The initial retrieval, BM25, has Recall@100 of 38.2% on average. CORPUSWALKER achieves 54.8% average Recall@100 over all 12 subsets. In comparison to the initial retrieval and retrieval-then-rerank pipeline, which has the same pool due to bounded recall, CORPUSWALKER outperforms by $+16.6$ points.

SlideGAR does adaptive selection from the corpus graph and initial retrieval results and ends up with having almost half of the document budget from initial retrieval. It reaches Recall@100 of 49.2%, and CORPUSWALKER gains over SlideGAR by $+5.6$ points (around 11%), which shows that graph-only-based exploration is not optimal. RGS, which starts the exploration with initial seeds and keeps exploring in the graph, is the strongest baseline. However, the exploration is mainly dependent on the ranking relevance, and the exploration is always tied to the query. Therefore, the pool quality of RGS is almost the same as SlideGAR (50.1% and 49.2%). CORPUSWALKER is 4.7 points above it and has the highest pool recall in all three domains of BRIGHT. The margin over RGS is highest on Coding ($+6.0$) and the theorem-based subsets ($+5.9$), and lowest on StackExchange ($+3.8$). Particularly, CORPUSWALKER outperforms RGS on AoPS by 45% and on TheoremQA-theorem by 35%. These gains clearly show the reasoning guided traversal utilizes the underlying query intent and explores both (lexical and semantic) signals encoded in the graphs. The reasoning traces can match the vocabulary of document features that are available at any traversal turn. We provide a concrete example in Figure 6 where the traversal agent chooses a lexical edge to find the relevant documents.

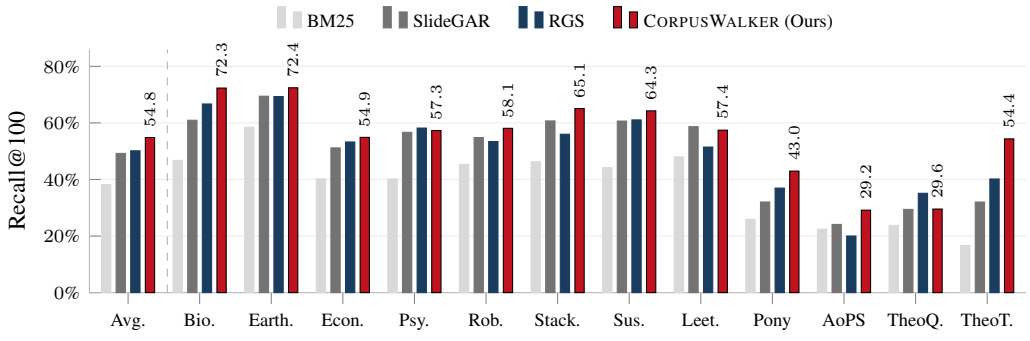


Figure 2: Recall@100 comparison. All methods retrieve 100 documents.

5.2 RANKING QUALITY

The objective of the graph traversal agent is to find relevant documents, however, it might add relevant document is last turns and simply keep adding them to the end of the retrieval list, as shown in Figure 4. So it applies ranking over the retrieved results using one LLM call, as described in Section 3.3. We report the effectiveness measured by nDCG@10 and Recall@10 in Table 1.

In comparison to the retrieval methods, BM25 and ReasonIR, CORPUSWALKER outperforms them by 22.1 and 11.9 points in nDCG@10 and by 22.1 and 10.7 points in Recall@10, respectively. In comparison to the retrieve-then-rerank pipelines, CORPUSWALKER outperforms the strongest baseline TFRank (which is a fine-tuned model for reranking for the BRIGHT benchmark) by 11 points in nDCG@10 and by 12.8 points in Recall@10. We observe similar gains in the listwise reranking, RankGPT, where CORPUSWALKER improves ranking and retrieval by 7.7 and 10.7 points.

Table 1: nDCG@10 and Recall@10 across BRIGHT subsets.

	StackExchange							Coding		Theorem-based			Avg.
	Bio.	Earth.	Econ.	Psy.	Rob.	Stack.	Sus.	Leet.	Pony	AoPS	TheoQ.	TheoT.	
nDCG@10													
Retrieval													
BM25	17.2	25.6	14.0	12.7	14.9	17.4	12.6	24.3	8.8	6.4	9.6	4.9	14.0
ReasonIR-8B	26.3	31.4	23.2	30.2	18.0	23.9	20.7	34.7	10.3	14.6	30.2	27.1	24.2
Retrieve-then-rerank (BM25 top-100)													
TFRank-8B	35.8	45.8	22.7	27.2	23.7	27.9	30.2	30.4	24.8	3.8	16.0	12.7	25.1
Qwen3-8B-Reranker	28.5	33.6	20.7	25.8	29.2	26.7	23.6	26.7	18.7	7.0	20.0	12.2	22.7
Rank1-7B	34.0	33.8	17.9	25.6	16.7	20.2	24.0	14.8	25.9	7.4	18.2	13.2	21.0
REARank-7B	30.2	35.1	21.5	26.5	24.0	24.0	26.5	25.5	22.1	7.8	17.5	9.1	22.5
RankGPT	36.6	41.1	24.7	35.0	31.9	31.2	33.2	33.7	28.6	9.7	21.3	13.7	28.4
Graph-exploration													
SlideGAR	38.7	42.7	28.6	42.4	31.9	30.8	38.3	34.9	20.8	7.9	22.3	22.4	30.1
RGS	43.2	**48.4**	**34.7**	**42.7**	33.1	29.9	38.0	33.3	23.9	8.8	**27.0**	32.6	33.0
CORPUSWALKER (Ours)	**46.9**	47.9	34.4	39.9	**39.6**	**35.3**	**41.2**	**35.3**	**29.9**	**18.7**	24.1	**39.8**	**36.1**
Recall@10													
Retrieval													
BM25	20.1	30.3	16.0	17.5	19.8	20.6	21.8	29.6	4.7	6.7	12.3	7.0	17.2
ReasonIR-8B	32.9	36.3	24.8	35.5	21.2	33.1	26.2	40.5	5.8	16.7	33.5	36.9	28.6
Retrieve-then-rerank (BM25 top-100)													
TFRank-8B	38.0	46.3	25.2	29.2	27.9	32.5	35.4	36.7	11.6	4.3	18.5	12.9	26.5
Qwen3-8B-Reranker	32.5	38.9	23.0	28.5	33.2	33.6	28.2	29.2	7.7	7.1	20.8	15.6	24.8
Rank1-7B	35.3	38.2	18.9	28.8	18.2	24.7	28.4	22.3	10.8	7.6	19.8	16.2	22.4
REARank-7B	33.7	37.6	20.7	27.8	27.8	29.2	29.8	32.0	9.4	8.7	18.8	11.2	23.9
RankGPT	38.9	42.3	24.0	33.7	33.6	35.8	35.4	39.9	14.1	9.8	21.5	14.3	28.6
Graph-exploration													
SlideGAR	43.4	41.5	29.4	43.6	33.6	35.0	42.2	40.8	9.2	6.6	24.6	24.7	31.2
RGS	45.4	51.2	35.8	**46.8**	37.5	36.6	43.3	39.1	11.3	8.1	**29.8**	36.5	35.1
CORPUSWALKER (Ours)	**55.1**	**52.9**	**36.1**	43.3	**44.1**	**44.1**	**46.6**	**42.7**	**14.3**	**18.7**	27.1	**47.0**	**39.3**

Graph traversal baselines, SlideGAR and RGS, explore a corpus graph and reach 30.1 and 33.0. These baselines decide *where* to expand based on the reranker's relevance, so a document is expanded because the reranker scored it highly, not due to judgment over whether its neighbours are worth the budget. Also, they utilize only a single type of edge relationship in the graph. In comparison to SlideGAR, CORPUSWALKER gains 6.0 nDCG@10 and 8.1 Recall@10 and outperforms on 11 of 12 subsets on both. In comparison to RGS, CORPUSWALKER gains 3.1 nDCG@10 and 4.2 Recall@10 and wins 8 and 10 of 12 subsets. The gains are larger: +9.9 nDCG@10 on AoPS, +7.2 on TheoremQA-theorems, +6.5 on Robotics, and +6.0 on Pony. RGS itself gains 2.9 nDCG@10 over SlideGAR from a pool of about the same recall, so its advantage is ranking. CORPUSWALKER's advantage over RGS comes from both a better pool (+4.7 Recall@100) and a better ranking of it. On average, CORPUSWALKER outperforms RGS by 3.1 points in nDCG@10 and noticable gains on AoPS (+9.9), Robotics (+6.5), Pony (+6.0), and Stackoverflow (+5.4). CORPUSWALKER also outperforms RGS by 4.2 points on average Recall@10.

5.3 GENERALIZATION TO FRESHSTACK

CORPUSWALKER (ours), generalizes to the FreshStack benchmark, where queries are StackOverflow questions over the code, documentation, and issues of software projects (LangChain, YOLO, Laravel, Angular, Godot), and each query is judged against a set of *nuggets*, the atomic facts its accepted answer states. We also report the standard retrieval metric Recall@50. CORPUSWALKER has the best average on all three metrics (41.3 α-nDCG@10, 67.8 Coverage@20, 44.7 Recall@50) as shown in Table 2. Each system reranks 100 documents. CORPUSWALKER outperforms the listwise reranking pipeline (denoted as RankGPT) and BM25 top 100 on every topic and every metric by 4.0 α-nDCG@10 and 6.8 Coverage@20. The reasoning aligns with the BRIGHT benchmark.

The reranker is bounded by the recall of documents returned by the first stage. However, we observe CORPUSWALKER provides proportional gains even in such cases. The reranker therefore provides most gains in regimes where BM25 has a good pool from first-stage retrieval (YOLO, Laravel) and least where it is not: on Godot it adds 16 points of α-nDCG@10 over BM25 while CORPUSWALKER adds 25, because the reasoning-guided traversal reaches the code and documentation chunks that the query's words do not, and the ranker surfaces them to top positions.

Table 2: FreshStack at budget of 100 documents/query. α-nDCG@10 (αN@10) & Coverage@20 (C@20) are FreshStack's official nugget-level metrics, Recall@50 (R@50) is document-level.

Method	LangChain			Yolo v7 & v8			Laravel 10 & 11			Angular 16, 17 & 18			Godot4			Avg.		
	αN@10	C@20	R@50	αN@10	C@20	R@50	αN@10	C@20	R@50	αN@10	C@20	R@50	αN@10	C@20	R@50	αN@10	C@20	R@50
BM25	19.0	42.9	22.3	16.2	37.8	34.7	28.7	60.8	40.9	24.2	54.0	31.4	12.2	31.1	26.6	20.1	45.3	31.2
RankGPT	30.8	56.2	26.4	38.2	63.2	44.1	47.7	73.6	47.9	41.7	67.8	38.1	28.2	44.2	30.5	37.3	61.0	37.4
SlideGAR	33.4	53.1	26.1	41.1	63.7	39.6	48.1	65.6	44.4	**42.8**	61.2	36.2	32.4	49.9	31.4	39.6	58.7	35.5
RGS	32.4	**58.6**	**30.7**	42.7	67.2	49.0	48.0	74.5	55.4	40.5	67.7	40.8	32.7	56.9	40.2	39.3	65.0	43.2
CORPUSWALKER	**34.3**	58.3	30.6	**43.7**	**73.6**	**50.4**	**48.3**	**78.1**	**55.5**	42.6	**67.9**	**41.0**	**37.6**	**61.2**	**45.9**	**41.3**	**67.8**	**44.7**

SlideGAR and RGS also start with the BM25 seeds and traverse over a corpus graph, and both end with pools with a similar number of relevant documents; Recall@100 is 50.1, 50.0, and 51 for SlideGAR, RGS, and CORPUSWALKER respectively. However, FreshStack rewards documents that are different nuggets in the answer. The baselines expand whatever their reranker currently scores highest, and since relevant documents neighbour one another in the graph, that keeps mining the neighbourhood of a nugget already found and adds its other supporting documents; the pool grows in redundancy more than in coverage. CORPUSWALKER decides where to expand by reading the summaries of its pool under a definition that rewards documents supporting *different* facts, so its pool covers more nuggets with fewer documents per nugget (1.53 vs. 1.64). Against SlideGAR, this shows as a large gain on the two coverage-style metrics, +9.1 Coverage@20 and +9.2 Recall@50 on average, with a smaller α-nDCG@10 gain (+1.7): SlideGAR's window-based reranking also drops many covered nuggets out of the window w, where the Meta Reranker reads the whole pool along with reasoning traces and still ranks them in the top-20.

In comparison to RGS, CORPUSWALKER outperforms on all three metrics (+2.0, +2.8, +1.5) and on most subsets of FreshStack. The **largest gains** (6.4 **points**) in Coverage@20 can be observed in Yolo. This makes our argument that relevance is not enough for graph exploration solid. Moreover, RGS reranks raw text at each step and makes more LLM calls per query, whereas CORPUSWALKER traversal calls over summaries and one ranking call. The token cost analysis is in Section 5.5.

5.4 TRAVERSAL REASONING EFFECT ON RANKING

The traversal agent generates the reasoning traces for why a particular document was expanded. In contrast to other graph expansion-based methods, we utilize the actions taken during traversal in the final ranking step. Our Meta Reranker reads the traversal reasoning traces and compressed document information that is represented in summaries and keywords. In this experiment, we answer RQ II and quantify whether the gains are coming from compressed information only or the traversal path taken by the agent. To this end, we apply standard listwise ranking on the pool only, which has summaries and keywords for each document. We compare this setup with our Meta Reranker, which also reads the traversal traces. We report the nDCG@10 performance on BRIGHT in Table 3. We clearly observe that for each subset of BRIGHT, ranking degrades with summaries-only input. The largest drops are on Pony of 5.2 points and the least on Stackoverflow of 0.2 points. The average drop on all 12 subsets of BRIGHT is 3.3 points. These performance drops highlight the role of our Meta Reranker and reasoning traces during traversal.

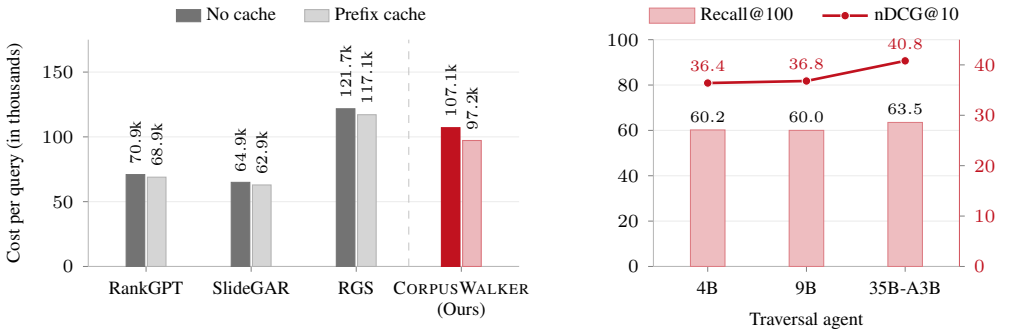
Rerankers like Rank1 Weller et al. (2025b) do relevance assessment at test time. However, they require a high compute budget, both tokens and latency, and output long reasoning traces. But the efficient ranking using CORPUSWALKER needs a low compute budget during reranking since it simply outputs the final ranking list. We also compare the effectiveness of CORPUSWALKER with methods that do reasoning at query level. For baselines, we use official CoT-style queries released by BRIGHT as well as recently proposed methods such as TongSearch-QR and ThinkQE, which use retrieved documents as feedback and refine queries iteratively. Diver-QExpand and ReDI fine-tune LLMs for query refinement. We report nDCG@10 performance on the StackExchange subset of BRIGHT in Table 4. CORPUSWALKER outperforms the strongest baseline by 2.5 points.

5.5 COMPUTATIONAL COST

In this section, we answer RQ III, and we compare the computational cost of baselines and our method. For this, we calculate the average token cost per query on BRIGHT. We define total token cost = input tokens + 4×output tokens (the output-to-input price ratio of widely used commercial models such as GPT-4o (OpenAI, 2026)). The dark bars in Figure 3a show the token costs (without

Table 3: nDCG@10 of Meta Reranker on same pool, with and without traversal reasoning.

	StackExchange							Coding		Theorem-based			Avg.
	Bio.	Earth.	Econ.	Psy.	Rob.	Stack.	Sus.	Leet.	Pony	AoPS	TheoQ.	TheoT.	
Summaries only	42.0	43.2	31.2	39.5	36.1	35.0	37.8	31.4	24.7	13.7	21.9	37.6	32.8
CorpusWalker (Ours)	**46.9**	**47.9**	**34.4**	**39.9**	**39.6**	**35.3**	**41.2**	**35.3**	**29.9**	**18.7**	**24.1**	**39.8**	**36.1**
Δ	+4.9	+4.7	+3.2	+0.4	+3.6	+0.2	+3.4	+4.0	+5.2	+5.0	+2.2	+2.2	+3.3



(a) Cost in terms of Token counts per query

(b) Traversal agent size

Figure 3: (a) Per-query cost on BRIGHT. (b) Agent size effect on the StackExchange subsets.

considering the cache tokens which repeat over iterations). The light bars give the same cost on a server with a prefix cache, i.e., prompt tokens that repeat the beginning of the previous call's prompt are not recomputed and are counted at 0.1 of a normal prompt token. CorpusWalker reads only summaries during traversal but outputs relatively more tokens than baselines. However, the ranking is done in an LLM call, the total number of LLM calls is $T + 1$, where T is the number of turns. On the BRIGHT benchmark, our traversal uses 9.1 turns to reach the budget of 100 documents, along with 1 LLM call for ranking (a total of 10.1 calls). On average, to fulfil the budget of 100 documents, RGS makes around 20 LLM calls, which is almost $2\times$ that of CorpusWalker. Also, RGS needs to keep the documents shortlisted in order to explore the graph. Therefore, at each turn, RGS ranks all documents it has found so far, and this ranking happens over the raw text of documents. Also, our Meta Ranker uses summaries. RGS costs 122k tokens (where 115.0k input and 1.7k output) per query, whereas our method only costs 107k (where 82.2k input and 6.2k output) and gains 3.1 nDCG@10 over RGS. Comparatively, listwise reranking and SlideGAR use fewer LLM calls and hence fewer tokens (71k and 65k respectively) but score 6 to 8 nDCG@10 points lower.

Finally, we observe how CorpusWalker behaves with the size of the traversal agent. We vary the traversal agent by Qwen3.5-4B and Qwen3.5-9B and our default Qwen3.6-35B-A3B (only 3B are active parameters), but we keep the Meta Ranker fixed. We report pool and ranking quality in Figure 3b. As we increase the traversal size from 4B to 9B, the pool (Recall@100) and ranking quality remain in the ballpark. As we go to a better traversal agent, the performance improves, Recall@100 by about 3.5 points and nDCG@10 by about 4 on StackExchange subsets of BRIGHT.

6 Conclusion

In this work, we introduce CorpusWalker, which employs graph exploration to improve search. Our approach casts graph traversal as a reasoning task, unlike existing graph-based adaptive ranking methods. Existing approaches employ heuristic traversal or are bounded by the re-ranker's notion of semantic relevance, which does not work well for reasoning-intensive retrieval tasks. They also employ a single fixed graph (lexical or semantic) and cannot adaptively determine the best graph to employ. CorpusWalker decouples ranking and graph traversal by employing a traversal agent that dynamically determines the graph type to employ for expanding documents and also adaptively decides the document to expand. Evaluating on reasoning-intensive retrieval benchmarks demonstrates consistent gains in retrieval and ranking performance. We also observe that though some inference cost is inferred by CorpusWalker, it is still sample-efficient minimizing number of re-ranker calls due to our one pass meta-ranking and is also cost-efficient compared to competitive baselines like RGS.

AI USE STATEMENT

We use open-source Qwen3.6-35B-A3B (as a generative model) as a backbone for baselines and our method, and the same model is used for generating document summaries offline, as described in Section 3. We used the generative AI tool (Claude model) to write LaTeX code for bar plots and figures. We also used a generative model to rephrase or fix grammatical mistakes in parts of the text. We used a generative AI tool to rewrite and restructure system prompts. We have not used generative AI tools to generate synthetic data sets, help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, or assist with translation. We used Claude-code to clean and restructure our repository. We take complete responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We release code along with our submission. We also release run files in TREC format to reproduce our results. We provide details on indexing, graph creation, hyperparameters (Section D), and all system prompts used in Section G.

REFERENCES

Akari Asai, Timo Schick, Patrick Lewis, Xilun Chen, Gautier Izacard, Sebastian Riedel, Hannaneh Hajishirzi, and Wen-tau Yih. Task-aware retrieval with instructions. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 3650–3675, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.225. URL https://aclanthology.org/2023.findings-acl.225/.

Yongqi Fan, Xiaoyang Chen, Dezhi Ye, Jie Liu, Haijin Liang, Jin Ma, Ben He, Yingfei Sun, and Tong Ruan. Tfrank: Think-free reasoning enables practical pointwise LLM ranking. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor (eds.), *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pp. 21020–21028. AAAI Press, 2026. doi: 10.1609/AAAI.V40I25.39244. URL https://doi.org/10.1609/aaai.v40i25.39244.

Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. Splade: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '21, pp. 2288–2292, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380379. doi: 10.1145/3404835.3463098. URL https://doi.org/10.1145/3404835.3463098.

Revanth Gangi Reddy, JaeHyeok Doo, Yifei Xu, Md Arafat Sultan, Deevya Swain, Avirup Sil, and Heng Ji. FIRST: Faster improved listwise reranking with single token decoding. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8642–8652, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.491. URL https://aclanthology.org/2024.emnlp-main.491/.

Nilesh Gupta, Wei-Cheng Chang, Ngot Bui, Cho-Jui Hsieh, and Inderjit S Dhillon. LLM-guided hierarchical search for end-to-end reasoning intensive retrieval. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=49vX1GYy4s.

Shizhe He, Avanika Narayan, Ishan S. Khare, Scott Linderman, Christopher Re, and Dan Biderman. An information theoretic perspective on agentic system design. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=isFHz8qf20.

Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Paper.pdf.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *CoRR*, abs/2503.09516, 2025. doi: 10.48550/ARXIV.2503.09516. URL https://doi.org/10.48550/arXiv.2503.09516.

Jongho Kim, Jaeyoung Kim, Jihyuk Kim, Yu Jin Kim, Seung-won Hwang, and Moontae Lee. Adaptive retrieval for reasoning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 37366–37381, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. doi: 10.18653/v1/2026.acl-long.1734. URL https://aclanthology.org/2026.acl-long.1734/.

Yibin Lei, Tao Shen, and Andrew Yates. ThinkQE: Query expansion via an evolving thinking process. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 17772–17781, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.965. URL https://aclanthology.org/2025.findings-emnlp.965/.

Chaofan Li, Minghao Qin, Shitao Xiao, Jianlyu Chen, Kun Luo, Defu Lian, Yingxia Shao, and Zheng Liu. Making text embedders few-shot learners. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025a. URL https://openreview.net/forum?id=wfLuiDjQ0u.

Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, EMNLP 2025, Suzhou, China, November 4-9, 2025*, pp. 5420–5438. Association for Computational Linguistics, 2025b. doi: 10.18653/V1/2025.EMNLP-MAIN.276. URL https://doi.org/10.18653/v1/2025.emnlp-main.276.

Zhuofeng Li, Haoxiang Zhang, Cong Wei, Pan Lu, Ping Nie, Yi Lu, Yuyang Bai, Shangbin Feng, Hangxiao Zhu, Ming Zhong, Yuyu Zhang, Jianwen Xie, Yejin Choi, James Zou, Jiawei Han, Wenhu Chen, Jimmy Lin, Dongfu Jiang, and Yu Zhang. Beyond semantic similarity: Rethinking retrieval for agentic search via direct corpus interaction. *CoRR*, abs/2605.05242, 2026. doi: 10.48550/ARXIV.2605.05242. URL https://doi.org/10.48550/arXiv.2605.05242.

Meixiu Long, Duolin Sun, Dan Yang, Junjie Wang, Yue Shen, Jian Wang, Peng Wei, Jinjie Gu, and Jiahai Wang. DIVER: A multi-stage approach for reasoning-intensive information retrieval. *CoRR*, abs/2508.07995, 2025. doi: 10.48550/ARXIV.2508.07995. URL https://doi.org/10.48550/arXiv.2508.07995.

Xueguang Ma, Liang Wang, Nan Yang, Furu Wei, and Jimmy Lin. Fine-tuning llama for multi-stage text retrieval. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, pp. 2421–2425, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400704314. doi: 10.1145/3626772.3657951. URL https://doi.org/10.1145/3626772.3657951.

Sean MacAvaney, Nicola Tonellotto, and Craig Macdonald. Adaptive re-ranking with a corpus graph. In Mohammad Al Hasan and Li Xiong (eds.), *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, Atlanta, GA, USA, October 17-21, 2022*, pp. 1491–1500. ACM, 2022. doi: 10.1145/3511808.3557231. URL https://doi.org/10.1145/3511808.3557231.

Niklas Muennighoff, Hongjin SU, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. Generative representational instruction tuning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=BC4lIvfSzv.

OpenAI. API pricing. https://developers.openai.com/api/docs/pricing, 2026.

June Park, Jun Kwon, Jonghyo Kim, and Jongwuk Lee. Group-aware adaptive retrieval for evidence navigation. *arXiv preprint arXiv:2609.02188*, 2026.

Xubo Qin, Jun Bai, Jiaqi Li, Zixia Jia, and Zilong Zheng. Reinforced query reasoners for reasoning-intensive retrieval tasks. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21250–21263, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1078. URL https://aclanthology.org/2025.emnlp-main.1078/.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Mandeep Rathee, Sean MacAvaney, and Avishek Anand. Guiding retrieval using llm-based listwise rankers. In Claudia Hauff, Craig Macdonald, Dietmar Jannach, Gabriella Kazai, Franco Maria Nardini, Fabio Pinelli, Fabrizio Silvestri, and Nicola Tonellotto (eds.), *Advances in Information Retrieval*, pp. 230–246, Cham, 2025. Springer Nature Switzerland. ISBN 978-3-031-88708-6.

Mandeep Rathee, Venktesh V, Sean MacAvaney, and Avishek Anand. Reproducing adaptive reranking for reasoning-intensive ir. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '26, pp. 3039–3049, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400725999. doi: 10.1145/3805712.3808568. URL https://doi.org/10.1145/3805712.3808568.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, apr 2009. ISSN 1554-0669. doi: 10.1561/1500000019. URL https://doi.org/10.1561/1500000019.

Rulin Shao, Rui Qiao, Varsha Kishore, Niklas Muennighoff, Xi Victoria Lin, Daniela Rus, Bryan Kian Hsiang Low, Sewon Min, Wen tau Yih, Pang Wei Koh, and Luke Zettlemoyer. ReasonIR: Training retrievers for reasoning tasks. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=kkBCNLMbGj.

Huatong Song, Jinhao Jiang, Yingqian Min, Jie Chen, Zhipeng Chen, Wayne Xin Zhao, Lei Fang, and Ji-Rong Wen. R1-searcher: Incentivizing the search capability in llms via reinforcement learning. *CoRR*, abs/2503.05592, 2025. doi: 10.48550/ARXIV.2503.05592. URL https://doi.org/10.48550/arXiv.2503.05592.

Hongjin SU, Howard Yen, Mengzhou Xia, Weijia Shi, Niklas Muennighoff, Han yu Wang, Liu Haisu, Quan Shi, Zachary S Siegel, Michael Tang, Ruoxi Sun, Jinsung Yoon, Sercan O Arik, Danqi Chen, and Tao Yu. BRIGHT: A realistic and challenging benchmark for reasoning-intensive retrieval. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ykuc5q381b.

Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. Is ChatGPT good at search? investigating large language models as re-ranking agents. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 14918–14937, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.923. URL https://aclanthology.org/2023.emnlp-main.923/.

Nandan Thakur, Jimmy Lin, Sam Havens, Michael Carbin, Omar Khattab, and Andrew Drozdov. Freshstack: Building realistic benchmarks for evaluating retrieval on technical documents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=54TTgXlS2U.

Runchu Tian, Xueqiang Xu, Bowen Jin, SeongKu Kang, and Jiawei Han. Corank: Llm-based compact reranking with document features for scientific retrieval. In Won Kim, Jae-Gil Lee, Kyuseok Shim, James Caverlee, David F. Gleich, Xiangliang Zhang, Qi He, Faisal Farooq, Rómer Rosales, Sayan Ranu, Haixun Wang, Yan Liu, Carl Yang, Leman Akoglu, Nitesh V. Chawla, and Vagelis Papalexakis (eds.), *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2, KDD 2026, Jeju Island, Korea, August 9-13, 2026*, pp. 12114–12125. ACM, 2026. doi: 10.1145/3770855.3818844. URL https://doi.org/10.1145/3770855.3818844.

Lidan Wang, Jimmy Lin, and Donald Metzler. A cascade ranking model for efficient ranked retrieval. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, pp. 105–114, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450307574. doi: 10.1145/2009916.2009934. URL https://doi.org/10.1145/2009916.2009934.

Orion Weller, Benjamin Chang, Sean MacAvaney, Kyle Lo, Arman Cohan, Benjamin Van Durme, Dawn Lawrie, and Luca Soldaini. FollowIR: Evaluating and teaching information retrieval models to follow instructions. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 11926–11942, Albuquerque, New Mexico, April 2025a. Association for Computational Linguistics. ISBN 979-8-89176-189-6. doi: 10.18653/v1/2025.naacl-long.597. URL https://aclanthology.org/2025.naacl-long.597/.

Orion Weller, Kathryn Ricci, Eugene Yang, Andrew Yates, Dawn Lawrie, and Benjamin Van Durme. Rank1: Test-time compute for reranking in information retrieval. In *Second Conference on Language Modeling*, 2025b. URL https://openreview.net/forum?id=Pg0PAvbhGv.

Haike Xu and Tong Chen. Beyond sequential reranking: Reranker-guided search improves reasoning intensive retrieval. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 113703–113717, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/b8c194e385f1d116710e24c30315dd2f-Paper-Conference.pdf.

Le Zhang, Bo Wang, Xipeng Qiu, Siva Reddy, and Aishwarya Agrawal. REARANK: Reasoning reranking agent via reinforcement learning. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 2458–2471, Suzhou, China, November 2025a. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.125. URL https://aclanthology.org/2025.emnlp-main.125/.

Yanzhao Zhang, Mingxin Li, Dingkun Long, Xin Zhang, Huan Lin, Baosong Yang, Pengjun Xie, An Yang, Dayiheng Liu, Junyang Lin, Fei Huang, and Jingren Zhou. Qwen3 embedding: Advancing text embedding and reranking through foundation models. *CoRR*, abs/2506.05176, 2025b. doi: 10.48550/ARXIV.2506.05176. URL https://doi.org/10.48550/arXiv.2506.05176.

Yunfei Zhong, Jun Yang, Yixing Fan, Lixin Su, Maarten de Rijke, Ruqing Zhang, and Xueqi Cheng. Reason to retrieve: Enhancing query understanding through decomposition and interpretation. *arXiv preprint arXiv:2509.06544*, 2025.

Shengyao Zhuang, Xueguang Ma, Zheng Yao, Shuai Wang, Bevan Koopman, Jimmy Lin, and Guido Zuccon. Rank-r1: Enhancing reasoning in llm-based document rerankers via reinforcement learning. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '26, pp. 4419–4425, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400725999. doi: 10.1145/3805712.3809961. URL https://doi.org/10.1145/3805712.3809961.

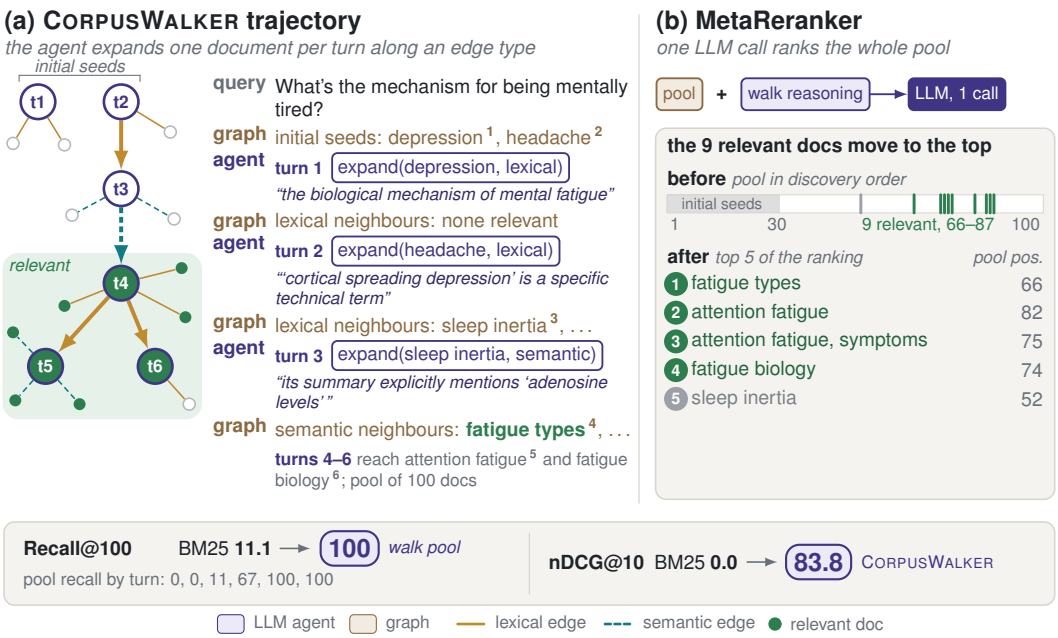


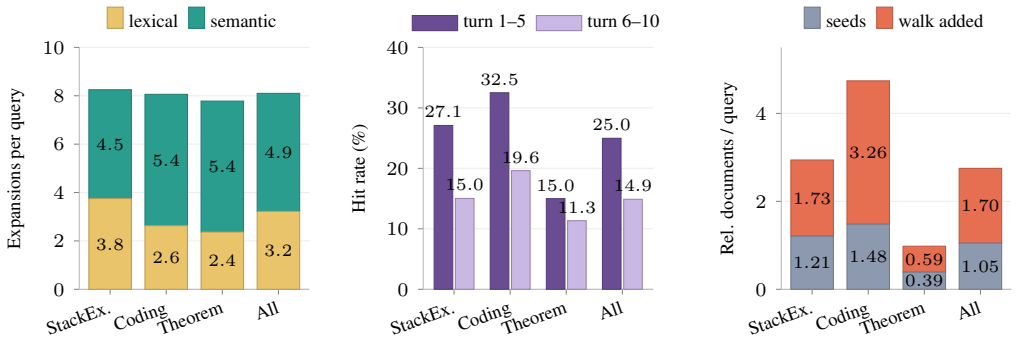
Figure 4: CorpusWalker on a BRIGHT biology query about mental fatigue. **(a)** The agent expands one document per turn and states why it picks a lexical or semantic edge. Two non-relevant hops (*headache*, *sleep inertia*) lead into the relevant region, and pool recall reaches 100% (BM25 recall@100 is 11.1%). **(b)** The relevant documents are found late (positions 66–87); one Meta Reranker call ranks them to the top (nDCG@10 goes from 0.0 to 83.8).

A Overview of CorpusWalker

We show the reasoning traces during traversal taken by CorpusWalker in Figure 4 (a) with a BRIGHT biology query. The traversal agent starts with initial seeds where none of the documents are relevant, and our traversal agent selects an initial seed document that talks about *depression* and then, in the 2nd turn, also expands the initial seed document about *headache*. After the 3rd turn, the traversal agent finds the document about *sleep inertia* which is non-relevant to the query, and expands it using a semantic edge in the 4th turn, where neighborhood documents share similar concepts such as *fatigue types, attention fatigue, symptoms*. After the 5th turn, our traversal agent finds all relevant documents and achieves recall of 100% for this query. We also show the positions of the relevant documents found during traversal, i.e., at 52, 66, 74, 75, and 82. Since we can compute the recall at inference time, our stopping criterion is the document budget, which is 100. In (b), we show how Meta Reranker utilizes the documents found and the corresponding reasoning during traversal. The reasoning traces consist of entries with the schema $\langle \texttt{turn_no}, \texttt{doc_id}, \texttt{expansion_rationale} \rangle$. as shown in Meta Reranker's system prompt (in Figure 9). Since we compressed the representation of the documents, the pool of 100 documents and traversal reasoning are processed in 1 LLM call. Next, Meta Reranker ranks 4 relevant documents in the first 4 positions, and Meta Reranker achieves an nDCG@10 of 83.8%, whereas the initial retriever had nDCG@10 of 0.

B CorpusWalker's Walk Statistics

A walk from CorpusWalker makes 8.1 expansions per query on average (7.8–8.2 per block), which is below the maximum turns we allow, $T = 20$ turns. For the majority of queries, the walk ends because the 100-document budget is full. Each expansion is one call to the agent. However, since the agent is not fine-tuned for taking such actions, it might result in an invalid action, e.g., the response could not be parsed or could generate a document that was not a valid choice. Also, the metareranker take one call; for 10.1 LLM calls per query in total, the count used in the cost analysis.



(a) Edge type use (b) Hits during traversal (c) Contribution from the walk

Figure 5: How the walk of CORPUSWALKER (Ours) behaves on BRIGHT, averaged over the queries of each domain (StackExchange, Coding, Theorem-based) and over all 12 subsets. (a) Expansions per query by the edge followed. (b) Share of expansions that are hits, i.e. the expanded document or a neighbour it added is relevant, in the first five and the next five turns. (c) Relevant documents per query among the BM25 seeds and among the documents the walk added.

Figure 5 summarises the traversal behavior of CORPUSWALKER. We mainly study the contribution of different document expansion edge types, lexical and semantic (in Figure 5a), the hit rate during the traversal stage (in Figure 5b), and the contribution of traversal in adding further relevant documents (in Figure 5c). In these plots, we discard the invalid expansion turns, since the agent retakes the turn in such cases.

OnStackExchange, the agent follows both edges about equally often (3.8 lexical and 4.5 semantic expansions). On the Coding and Theorem-based subsets, it follows the semantic edge about twice as often as the lexical one (5.4 against 2.6 and 2.4). Over all of BRIGHT, it makes 3.2 lexical and 4.9 semantic expansions per query.

It is important to note that BRIGHT has sparse relevance, i.e., most subsets have 1.8–7.8 relevant documents per query, the exception being pony (19.8). We count an expansion as a *hit* if the expanded document is relevant or if at least one neighbour it added to the pool is relevant. Early expansions hit more often than later ones in every domain (Figure 5b). Over all of BRIGHT, 25.0% of expansions in turns 1–5 are hits, against 14.9% in turns 6–10. The agent expands its most promising documents first. Also, it might be the case that neighbors of the later turns are already added in the pool and are not counted as a *hit*. Hits are most common on Coding (32.5% early, 19.6% late) and least common on the Theorem-based subsets (15.0% and 11.3%), where each query has few relevant documents.

On average, the initial BM25 seeds contain 1.05 relevant documents per query, and the walk adds 1.70 more (Figure 5c). The walk adds the most on Coding, more than twice what the seeds contain (3.26 against 1.48). On StackExchange, it adds 1.73 to the 1.21 in the seeds. On the Theorem-based subsets, both numbers are small: the seeds contain only 0.39 relevant documents per query, and the walk adds 0.59.

C FURTHER QUALITATIVE STUDY OF CORPUSWALKER

Figure 6 explains agent traversal on a Maths (AoPS) query and a StackExchange (Biology) query. Figure 7 explains traversal actions per turn on a Laravel query from the FreshStack benchmark.

Maths (AoPS). *The graph of* $9x + 223y = 2007$ *is drawn on graph paper. How many* 1×1 *squares have interiors entirely below the graph and in the first quadrant?* Gold: two AoPS solutions that apply Pick's theorem (1998 AIME #2, 2013 AMC 12A #13). Neither is among the initial seeds.

Turn 0, problem analysis: "a lattice point counting problem, specifically related to Pick's Theorem or the geometry of numbers: find the intercepts, the area of the right triangle, then the number of lattice points."

- t0–t1 SEMANTIC → two problems about lattice points in bounded regions ("explicitly deals with lattice points and counting within a bounded region, the same core construct") – 25 documents, no gold. The pool now holds many lattice-point problems, none of which name the theorem.
- t2 SEMANTIC → "lattice points on the line $y = -2x + 18$ in the first quadrant" – 12 documents, including **1998 AIME #2**, whose summary reads "uses Pick's theorem to count ordered pairs… by calculating the area".
- t3 LEXICAL from 1998 AIME #2 – the agent now has the exact term: "its summary explicitly states it uses *Pick's theorem*… this directly references the core theorem identified" – 16 documents, including **2013 AMC 12A #13** ("Pick's theorem to calculate the area of a quadrilateral defined by lattice points"). Both gold documents are in the pool after four expansions.

StackExchange (Biology). *Can plants get cancer? I can't think of any reason why plants wouldn't be able to, but I've never heard of a plant growing a tumor.* Gold: three passages from the Wikipedia article *Agrobacterium tumefaciens* (the bacterium that causes crown-gall tumours). None among the seeds.

Turn 0, problem analysis: "establish the biological definition of cancer (uncontrolled cell division, neoplasia, loss of regulatory control) and determine whether plants possess the cellular machinery and regulatory pathways that, when they fail, produce such a state."

- t0 SEMANTIC → *Hyperplasia* §2, which "distinguishes hyperplasia from neoplasia: hyperplastic cells remain under normal regulatory control" – "the critical theoretical distinction needed to reason about whether plant growth abnormalities are cancer" – 12 documents, no gold.
- t1–t2 SEMANTIC → two *Meristem* passages (indeterminate growth; the CLAVATA–WUSCHEL feedback loop that regulates it) – "the established problem requires distinguishing true neoplasia from other uncontrolled growth in plants, such as pathogen-induced galls" – 20 documents, no gold. The agent has now built the frame: what regulates plant cell division, and what would count as its failure.
- t3 SEMANTIC → *Meristem* §6, "the induction of meristems in legume roots by Rhizobia bacteria" – "to complete the picture I need the specific pathology" – 7 documents, including the first gold passage, ***Agrobacterium tumefaciens* §0**: "causes crown gall disease by inserting T-DNA from a plasmid into the plant genome".
- t4 LEXICAL from that passage – "the most critical missing piece is the specific mechanism of plant tumours, particularly crown gall" – 16 documents, **the other two gold passages** of the same article. All three gold in five expansions.

Figure 6: What the two edge types contribute on BRIGHT. In both walks the SEMANTIC edge does the crossing – from "lattice points" to a solution that names Pick's theorem, from bacteria that induce meristems to the bacterium that induces tumours – and the LEXICAL edge does the harvesting once an exact term or a gold document is in hand, pulling in the documents that spell the same theorem or belong to the same article. Neither query's gold shares vocabulary with the query itself ("Pick", "Agrobacterium", "crown gall" appear nowhere in it); the agent's problem analysis supplies the concepts the semantic edge then follows.

D Reproducibility of CorpusWalker

D.1 Indexing and Graph Creation

We use PISA for sparse indexing and use a BM25 retriever (following the BRIGHT official implementation, we set $k_1 = 0.9$ and $b = 0.4$). We use ReasonIR-8B [1] as a dense retriever. For kNN corpus graphs, we follow the official implementation from SlideGAR Rathee et al. (2025) and use a BM25 index for lexical graphs and a ReasonIR index for semantic graphs, and keep the number of

[1] https://huggingface.co/reasonir/ReasonIR-8B

Query (Laravel). *Large store database design in Laravel. My store products are clothes, shoes, pants, laptops, phones, headphones… I am confused how to develop my products database* [schema shown as an image].

Nuggets of the accepted answer (7; the answer is a schema redesign, not a single fact): N1 use a many-to-many relationship for product options; N2 drop the per-type columns `color_id`, `size_id`; N3 rename `products_options` to `variations`; N4 drop the `colors` and `sizes` tables; N5 one `options` table with a generic `type` column; N6 a pivot table `options_variations`; N7 any option type can then attach to any product. **The initial seeds cover none of them.**

The walk (8 expansions; each line is the document the agent chose, the reason it gave, and the nuggets its neighbours added):

- t0 `docs/queries.md` (Query Builder: raw expressions and joins) — "the candidates are largely irrelevant (DB2, Solr, PHP history, Laravel marketing); `docs/queries.md` documents Laravel's Query Builder, which is essential for filtering products with variable attributes" → 12 documents, **N1, N6, N7**.
- t1 `docs/eloquent-relationships.md` (many-to-many with `belongsToMany`, pivot columns) — "the core technical challenge is choosing between EAV, JSON columns, or polymorphic relationships; I have already explored general Query Builder methods" → 11 documents, **N3, N4, N5**.
- t2–t4 polymorphic relationships (`morphToMany`), custom pivot models, attribute casting — the agent keeps refining the same design question ("the core issue is that the user's current schema … is inefficient") → 24 documents, no new nuggets.
- t5 `docs/migrations.md` (create, rename and drop tables with the Schema builder) — "the next critical piece is the database schema definition to implement the tables for these polymorphic or EAV structures" → 9 documents, **N2**.
- t6–t7 mass assignment, index and foreign-key migrations → 14 documents, no new nuggets.

All seven nuggets are covered after three productive expansions, each opening a different part of the answer: the relationship pattern, the Eloquent implementation, the migration that reshapes the tables. RGS, seeded from the same collection, covers none of the seven for this query.

Figure 7: How CORPUSWALKER assembles a multi-part answer on FreshStack. The agent reasons about what the answer must contain ("EAV, JSON columns, or polymorphic relationships"), and each expansion it chooses opens the neighbourhood of a different piece; the pool ends up covering distinct nuggets rather than several documents for the same one. The reasons are the agent's own, lightly shortened.

neighbors $k = 16$. For DiskANN Jayaram Subramanya et al. (2019) graphs, we follow the RGS Xu & Chen (2026) official implementation and keep the same number of neighbors. We release code for indexing and graph creation in our repository. Following the official setup, SlideGAR uses a kNN graph (using ReasonIR) and RGS uses a DiskANN graph (based on ReasonIR embeddings). CORPUSWALKER uses lexical graph using bm25 with 16 neighbours and semantic DiskANN graph using ReasonIR. Both lexical and semantic graphs are scalable to millions of documents (as used in SlideGAR), in contrast to other fine-grained graphs/tree structure methods like LATTICE Gupta et al. (2026).

All graphs are built offline and have small memory footprints. Stored with a 4-byte neighbour id and a 2-byte weight per edge, a 16-neighbour graph over BRIGHT corpora (1.15M documents) takes 110 MB, as do the lexical and the kNN graph; the DiskANN graph takes 107 MB, and 23.5 MB for the five FreshStack corpora (0.27M documents). The largest single graph, DiskANN over LeetCode's 414k documents, takes 41 MB. With 32 CPU threads, building the DiskANN graphs takes 13.7 min for all eleven BRIGHT corpora (from 3.2 s for Pony to 6.2 min for LeetCode) and 1.9 min for the five FreshStack corpora. The exact kNN graphs (lexical using BM25 index) used by SlideGAR take 25.5 min and 2.3 min on the same CPU.

Table 4: Query-level reasoning on the StackExchange subsets, nDCG@10. *Numbers are taken from ReDI Zhong et al. (2025).

Method	Query rewriting	Bio.	Earth.	Econ.	Psy.	Rob.	Stack.	Sus.	Avg.
Claude-3-opus CoT	Direct LLM	51.7	49.1	22.0	33.3	14.7	19.6	19.1	29.9
GPT-4 CoT	Direct LLM	52.1	49.9	18.5	33.5	13.0	21.9	19.9	29.9
TongSearch-QR (7B)*	Fine-tuned LLM	**57.9**	50.9	21.9	37.0	21.3	27.0	25.6	34.5
ThinkQE (14B)*	Iterative refinement	47.2	49.5	29.0	37.8	20.9	28.2	28.8	34.5
DIVER-QExpand (14B)*	Iterative refinement	50.9	50.3	22.4	35.3	22.2	27.3	26.4	33.5
ReDI (8B)*	Fine-tuned LLM	49.0	**53.5**	28.7	**43.4**	27.5	**36.3**	29.4	38.3
CorpusWalker (Ours)	–	46.9	47.9	**34.4**	39.9	**39.6**	35.3	**41.2**	**40.8**

Table 5: BRIGHT statistics. Relevant documents per query counts judgements with a positive label in the released relevance judgements, averaged over queries that have one; the excluded documents (label -100) are not counted. Query and document lengths are those reported by the benchmark (GPT-2 tokenizer). AoPS and TheoremQA-Questions share one corpus.

	StackExchange							Coding		Theorem-based		
	Bio.	**Earth.**	**Econ.**	**Psy.**	**Rob.**	**Stack.**	**Sus.**	**Leet.**	**Pony**	**AoPS**	**TheoQ.**	**TheoT.**
#Queries	103	116	103	101	101	117	108	142	112	111	194	76
#Documents	57,359	121,249	50,220	52,835	61,961	107,081	60,792	413,932	7,894	188,002	188,002	23,839
Rel. documents / query	3.6	5.0	7.8	6.9	5.1	4.1	5.3	1.8	19.8	4.7	3.2	2.0
Avg. query length	115.2	109.5	181.5	149.6	818.9	478.3	148.5	497.5	102.6	117.1	93.4	91.7
Avg. document length	83.6	132.6	120.2	118.2	121.0	704.7	107.9	482.6	98.3	250.5	250.5	354.8

D.2 Hyperparameters

All baselines used BM25 as the initial retriever in the retrieve-then-rerank setup and therefore use BM25 seeds for graph exploration. We use BM25 since it is efficient, training-free, and outperforms many dense retrievers, as shown in the BRIGHT paper SU et al. (2025). Also, it can be adapted to new benchmarks. All pipelines retrieve or rerank exactly 100 documents. For the sliding window-based listwise reranking baselines RankGPT, SlideGAR, and RGS, we set window size $w = 20$ and step size $b = 10$. For CorpusWalker, we use a seed size of 30. Since we use only compressed information, these documents can fit into a single window easily. This also reduces the number of turns for our traversal and makes our method efficient.

E Query-level Reasoning

In Table 4, we report the nDCG@10 performance on the StackExchange domain of BRIGHT for query-level reasoning methods.

F Datasets Statistics

We report the dataset statistics for BRIGHT in Table 5 and for FreshStack in Table 6.

G Prompts

Figure 8 shows the system prompt for our traversal agent. Figure 9 shows the system prompt for Meta Reranker. Finally, in Figure 10, we show the system prompt for our information compressor that generates summaries and keywords offline.

CorpusWalker traversal prompt

```
You are a retrieval agent exploring a corpus laid out as a graph.

GOAL: assemble the most complete set of documents satisfying the relevance
definition. Do not answer the query.

Expanding a document adds its NEIGHBOURS, not the document itself. Pick the
```

Table 6: FreshStack statistics. A document is a chunk and a query is its title and body; a nugget is one part of the answer, and relevance judgements are given per nugget. Relevant documents per query counts distinct documents with a positive label. Query length and the repository counts are those reported by the benchmark (`cl100k` tokenizer); the benchmark reports the length of the answer rather than of a document, so the document length is measured here with the same tokenizer.

	LangChain	**Yolo v7 & v8**	**Laravel 10 & 11**	**Angular 16, 17 & 18**	**Godot4**
#Queries	203	57	184	129	99
#Documents	49,514	27,207	52,351	117,288	25,482
Nuggets / query	3.1	3.5	3.0	3.2	3.3
Rel. documents / nugget	5.7	3.9	3.2	4.4	2.9
Rel. documents / query	10.9	7.4	6.0	8.7	5.9
Avg. query length	473.4	497.1	474.4	463.3	350.4
Avg. document length	1,060.4	1,252.3	776.5	732.5	1,089.1

```
document whose neighbourhood likely holds what you need, which is often not
the document that best matches the query.

Only the documents listed in HISTORY below have you actually looked at and
chosen to expand. CANDIDATES is everything currently sitting unexpanded in
your pool -- you have NOT visited these, and a document appearing there is
not "covered": it is raw material you have not yet acted on. Do not describe
CANDIDATES contents as already explored.

EDGES
  lexical  - joins documents sharing distinctive terms. Exact spelling only.
             Reaches a named target only if the source document's own
             keywords contain that term.
  semantic - joins documents close in embedding space. Crosses vocabulary
             gaps, drifts toward general topical similarity.

SUMMARY is authoritative. KEYWORDS are weak evidence: shared terms do not
make a document relevant if its summary describes a different technique.

QUERY: {query}
RELEVANCE DEFINITION: {relevance_definition}
BUDGET: {budget_status}
ESTABLISHED PROBLEM: {problem}
EDGE YIELD: {edge_yield}
  Counts new documents, not relevant ones. A high count can be topical
  drift. One input among several.
HISTORY (documents you have actually expanded so far -- nothing else counts
as visited):
{expansion_history}
CANDIDATES ({n_already_expanded} omitted, already expanded):
{pool_listing}
VALID CHOICES: {valid_docnos}

STEPS
1-2. If ESTABLISHED PROBLEM reads "(none)", work out what the query is
   really asking and what an answer would have to establish or invoke --
   name the mechanisms, theorems, algorithms or constructs explicitly,
   including ones the query never mentions. Otherwise it is settled: reply
   "SAME" and skip to 3.
3. What material satisfies "{relevance_definition}" here? Usually several
   documents, not one.
4. Compare your 2-3 strongest candidates on what sits NEXT TO each, using
   only HISTORY to judge what you have already mined. Pick one. Then decide
   the edge FROM THAT DOCUMENT by what your own comparison actually argued:
   - If you can name a SPECIFIC shared term or spelling that appears BOTH in
     your Step 2 analysis AND in this candidate's own KEYWORDS list, declare
     lexical and name that exact term in your reasoning.
   - If your comparison instead rests on shared meaning, concept, algorithm
     or structure -- with no such spelled term in common -- declare
     semantic. Do not declare lexical on the strength of a structural or
     conceptual match alone; that is a semantic argument regardless of which
     edge you meant to pick.

Reply with exactly this JSON, nothing else:
{"problem": "<Steps 1-2, or SAME>",
  "reasoning": "<Step 4 comparison: what sits next to each candidate, using
   HISTORY (not CANDIDATES) to judge what is already mined; then either the
   exact shared term justifying lexical, or the shared concept justifying
   semantic>",
```

```
  "expand": "<docno from VALID CHOICES>",
  "via": "<lexical|semantic>"}
```

Figure 8: CORPUSWALKER's traversal prompt (Section 3.2).

Meta Reanker prompt conditioned on traversal reasoning traces

```
You are ranking documents for a reasoning-intensive query. {relevance_definition}

QUERY:
{query}

An exploration agent has already searched a document graph for this query. It repeatedly
chose one document and pulled in that document's nearest neighbours. Its choices are
evidence you should use:
- A document the agent CHOSE TO EXPAND was judged promising by an agent that had read it.
Treat "expanded" as genuine positive evidence.
- Earlier expansions carry more weight than later ones.
- A document that entered the pool because some expanded parent pulled it in inherits
partial evidence from that parent.
- Seeds came from a keyword search and are often on-topic in wording while missing what the
query actually needs.

This evidence is a prior, not a verdict. Judge each document's own content against the
relevance definition above first.

THE AGENT'S OWN REASONING, in order (turn, the document it expanded, why, and how many new
documents that expansion contributed):
{trace_listing}

CANDIDATES (identifier, provenance, then the document's summary and keywords):
{candidate_listing}

Rank ALL {n} candidates from most to least relevant to the query's underlying reasoning.

Do NOT write any analysis, preamble or explanation -- reply with EXACTLY one line: the
identifiers in rank order, most relevant first, separated by " > ".
Use the identifiers exactly as given, e.g. [3] > [1] > [2]
```

Figure 9: The ranking prompt for Meta Reranker (Section 3.3).

System prompt for offline document feature extraction

```
Analyze the following document and extract compact features that will be shown to an agent
deciding whether this document (and its neighbours in a document graph) are worth exploring
further -- WITHOUT showing the agent the full text.

Document:
<document>
{document}
</document>

Provide:
1. A summary (1-2 sentences) of the document's SPECIFIC content -- name the concrete
technique, theorem, API/function, or finding it covers. Not a generic topic label (e.g. NOT
"this document discusses arithmetic" -- instead "explains Pony's checked-arithmetic
operators and their overflow behaviour").
2. 5-10 specific keywords: exact technique/API/theorem/function names and concrete terms
from the document -- not broad category words (e.g. NOT "programming", "math").

If the document is only navigation, headers/footers, or raw tabular/lookup data with no
explanation of any technique or concept, output the tags empty (just
<summary></summary><keywords></keywords>) and stop.

Output ONLY in this exact format, nothing else:
<summary>
The summary goes here.
</summary>
<keywords>
keyword1, keyword2, keyword3, ...
</keywords>
```

Figure 10: System prompt for offline document feature extraction (Section3.1).