PILLAR: PRIVATE INVERTED-INDEX LEXICAL LOOKUP FOR AUGMENTED RETRIEVAL

Anonymous authors
Paper under double-blind review

ABSTRACT

Retrieval-augmented generation (RAG) hands the user's query to whoever hosts the corpus. We propose PILLAR, a Privacy-Preserving RAG (PPRAG) system based on Private Information Retrieval (PIR) in which a client utilizes the k documents most similar to their query from a server-held and publicly known corpus to respond to their query, while the server learns nothing about the query, either its terms or its access pattern. Prior PPRAG constructions rely on dense retrieval alone, translating approximate nearest-neighbor search into many query-dependent rounds of PIR, and pay for it in both latency and retrieval quality. PILLAR instead performs *private hybrid retrieval* in two stages. A sparse stage issues a small, fixed number of PIR queries against a carefully designed index of precomputed BM25 scores, filtering the corpus down to candidates that share terms with the query without the server ever seeing which terms these are. A dense stage then fetches only those candidates' *document embeddings* and re-ranks them locally, avoiding the many costly PIR queries that private dense retrieval typically requires. We instantiate PILLAR with two protocols that trade latency against retrieval quality, each built on a different private rendering of lexical search. PILLAR-Bin bins posting lists into a hash table and is a single-round design that achieves lower latency than state-of-the-art private retrieval schemes. PILLAR-Tree turns block-max pruning into an oblivious tree traversal combined with cuckoo hash tables and achieves the highest retrieval quality at lower latency than state-of-the-art schemes.

1 INTRODUCTION

Retrieval-augmented generation (RAG) is now the standard way to ground a Large Language Model (LLM) in knowledge that is not in its parameters (Lewis et al., 2020; Gao et al., 2023), and it is the mechanism major search engines rely on for AI summaries. A RAG pipeline consists of a client that issues a query, a server that hosts a document corpus, and an LLM that generates an answer to the client's query by retrieving documents from the server to augment its generated response. This poses a severe privacy risk, the query is handed, verbatim, to the server. Exposing a client's query carries real risk; search engine logs can trivially be used to identify clients (Barbaro and Jr., 2006), clients disclose sensitive personal details to LLMs (Mireshghallah et al., 2024), and current models infer identifiable information from very little text (Staab et al., 2024).

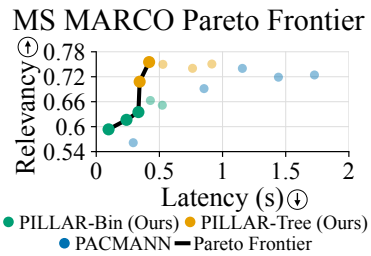


Figure 1: Pareto between a state-of-the-art private dense retrieval method PACMANN, and our private hybrid retrieval methods (PILLAR-Bin & PILLAR-Tree). In MS MARCO, our PILLAR-Tree (in orange) dominates across all quality metrics, while our PILLAR-Bin (in green) dominates in speed/latency.

Privacy-Preserving RAG. A privacy-preserving RAG (PPRAG) protocol answers a client's query over a public corpus such that the server hosting the corpus learns nothing about the query. Due to the public nature of the corpus, such protocols are typically built on Private Information Retrieval (PIR) (Chor et al., 1995), a cryptographic primitive that lets a client fetch a non-encrypted item from a server without revealing which item was fetched. Existing PPRAG work falls into two categories, *dense* and *sparse* retrieval, with the majority focusing on the former. Among dense retrieval protocols, Tiptoe (Henzinger et al.,

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.

2023) privately searches over clustered embeddings, Compass (Zhu et al., 2025) traverses an HNSW graph using Oblivious RAM (ORAM), and PACMANN (Zhou et al., 2024b), the current state-of-the-art (SOTA), has the client traverse an approximate nearest neighbor (ANN) graph. Sparse retrieval, by contrast, is represented by Coeus (Ahmad et al., 2021), which scores the query against a term-frequency matrix.

Limitations of Existing PPRAG. Existing PPRAG protocols face two limitations: First, ANN traversal is adaptive; each hop depends on the query, and the walk terminates once no closer neighbors are found. Hiding the access pattern therefore requires a fixed, query-independent schedule. On MS MARCO, PACMANN requires 20 round trips per query yet achieves only 0.266 MRR@10, compared to roughly 0.31 for a non-private ANN baseline over identical embeddings Zhou et al. (2024b). Second, existing protocols support only a single retrieval architecture, whereas *real RAG deployments* commonly use hybrid architectures, with lexical scoring followed by comparing learned embedding similarities (See Appendix G.2 for more details). The two architectures fail in complementary ways: learned embeddings encode word meanings using their training data but fail when a query uses the same word in a different context, while lexical scoring does not consider semantically relevant words (Ma et al., 2021; Thakur et al., 2021b; Bruch et al., 2023). Used together, the methods mitigate each other's weaknesses. This points to a gap in PPRAG literature:

> *No PPRAG uses both sparse and dense retrieval architectures, and the single-architecture alternatives pay for it in number of interactions and in answer quality.*

Contributions. We present PILLAR, a new *hybrid-retrieval*[1] PPRAG architecture that retains the strengths while limiting the weaknesses of previous dense or sparse PPRAG protocols. PILLAR achieves a "best-of-both-worlds" design, and we demonstrate its effectiveness with two new PPRAG protocols: `PILLAR-Bin` and `PILLAR-Tree`. `PILLAR-Bin` uses lexical scoring functions to resolve a query in a *single* round, achieving low latency. `PILLAR-Tree` adapts block-max BM25 Broder et al. (2003); Ding and Suel (2011) to perform multiple rounds of increasingly accurate retrievals, achieving unmatched quality compared to SOTA. We evaluate both the quality of the answer and latency for `PILLAR-Bin`, `PILLAR-Tree`, and the SOTA PACMANN, and provide an extensive evaluation in Section 5 and Appendix B.1. An indicative result is in Figure 1, which shows the *Pareto frontier* over the Top-5 configurations for each method. Specifically, it captures the trade-offs between the latency required to retrieve documents for a query (X-axis) and the Relevancy of an LLM-generated answer to that query, augmented with those documents (Y-axis). The *only configurations* on the frontier belong to either `PILLAR-Bin` or `PILLAR-Tree`, never once is PACMANN better in either latency or Relevancy. In summary, our core contributions are as follows:

- **Hybrid Retrieval Under PIR.** We give the first PPRAG protocols that combine lexical and semantic scoring, and prove that both `PILLAR-Bin` and `PILLAR-Tree` hide a client's query from a semi-honest server in the PIR-hybrid model.
- **Comprehensive Evaluation.** We benchmark both protocols against the SOTA, PACMANN, and plot the best configurations of all three protocols using a *Pareto frontier* in Section 5.2, finding that PILLAR accounts for 93% of the configurations on the frontier. We test on two standard RAG datasets, MS MARCO Bajaj et al. (2018) and SciFact Wadden et al. (2020), reporting retrieval quality, communication cost, and latency. Unlike PACMANN, which evaluates retrieval alone, we also evaluate the end-to-end answers generated from the retrieved documents, using the Retrieval Augmented Generation Assessment metrics *faithfulness* and *answer relevancy* Es et al. (2025).
- **A Low-Latency PPRAG Protocol.** The fastest PACMANN configuration needs 0.293 s, while `PILLAR-Bin` retrieves documents in as little as 0.098 s on MS MARCO, with comparable quality.
- **A High-Quality PPRAG Protocol.** The highest quality documents across the best configurations are retrieved by `PILLAR-Tree`, with answer Relevancy 0.75. The PACMANN configuration that finds the best documents achieves 0.72 answer relevancy but requires over $2\times$ more latency.

[1] In standard literature, a "hybrid-retrieval RAG protocol" refers to a protocol that combines the scores from both dense and sparse retrieval; here hybrid-retrieval refers strictly to the retrieval methods, not the final score

2 BACKGROUND & PRELIMINARIES

Notation. A *corpus* $\mathcal{C} = \{D_1, \ldots, D_n\}$ is an ordered set of n documents. The index i of a document $D_i \in \mathcal{C}$ is also referred to as a document's *identifier*. An *analyzer* $\mathsf{Analyze}$ maps raw text to a finite sequence of tokens called *terms*. The *vocabulary* $\mathcal{V} = \bigcup_{D \in \mathcal{C}} \mathsf{Analyze}(D)$ is the set of all unique terms in a corpus. An *embedding function* Emb maps raw text to a d-dimensional dense vector. With the term *document embedding* $e_D = \mathsf{Emb}(D) \in \mathbb{R}^d$, we refer to a document's dense vector representation. A *query* Q is raw text supplied by a client, so that $\mathsf{Analyze}(Q) = (w_1, \ldots, w_{|Q|})$ are its *query terms* and $e_Q = \mathsf{Emb}(Q) \in \mathbb{R}^d$ its *query embedding*. We sometimes abuse the notation Q to denote both a sequence of terms and the index of documents to be returned through PIR. We make this distinction clear by referring to the latter case as the *PIR query*.

Private Information Retrieval (PIR). There are multiple variants of PIR Chor et al. (1995); Kushilevitz and Ostrovsky (1997); Beimel et al. (2000) but we focus on Single-server PIR with preprocessing designs Zhou et al. (2024a); Corrigan-Gibbs and Kogan (2020); Beimel et al. (2000). These PIR protocols involve interactions between a stateful client `clnt` and a server `srv`, consisting of a (i) preprocessing phase followed by a (ii) query phase. In the PIR preprocessing phase, `srv`'s input is a publicly known $\mathcal{C}$, which it sends to `clnt`, who initializes a private state. During the query phase, `clnt`'s input is an index $0 \leq i \leq |\mathcal{C}|$ (w.r.t. to $\mathcal{C}$) and constructs a PIR query Q to retrieve the requested document $D_i \in \mathcal{C}$. `clnt` sends Q to `srv` and `clnt` uses the responses from `srv` and his internal state to output D_i. A PIR protocol is *secure* if `srv` does not discover anything about the client's input for the requested index i. Consistent with prior work Henzinger et al. (2023); Zhou et al. (2024b), we consider a semi-honest `srv`, where `srv` follows the protocol but tries to recover i.

Privacy-Preserving Retrieval Augmented Generation (PPRAG) Functionality. A standard PIR protocol is incompatible with PPRAG because a `clnt` takes query terms Q as input rather than an index i, and outputs the documents most similar to Q rather than the single document at index i, where similarity is measured by a function $\mathsf{score}(\cdot, \cdot)$ mapping a (query, document) pair to a score under some metric. For the cryptographic formalism, we consider the ideal two-party PPRAG functionality, where the server and client both have input, but only the client has output, defined by $\mathsf{PPRAG}(\mathcal{C}, Q) = \left(\perp, \mathcal{D}^k\right)$. In words: `srv` takes in a public corpus $\mathcal{C}$ and outputs nothing (denoted $\perp$), `clnt` takes as input a private Q and outputs k documents $\mathcal{D}^k \subseteq \mathcal{C}$ such that, for every $D \in \mathcal{D}^k$, and every $D' \in \mathcal{C} \setminus \mathcal{D}^k$ we have that: $\mathsf{score}(Q, D) \geq \mathsf{score}(Q, D')$. The new protocol PILLAR in this work realizes this ideal functionality of PPRAG.

3 CHALLENGES OF PIR UNDER RAG ARCHITECTURE

This section discusses sparse and dense retrieval architectures for RAG with a public corpus, with particular attention to the challenges that arise when scaling each to a PIR-based *private analog*.

3.1 SPARSE RETRIEVAL AND PRIVACY.

At a high level, *sparse retrieval* scores each document (via lexical scoring functions) by the query terms appearing in it verbatim, weighting rare terms more heavily than common ones, and returns the k highest-scoring documents. Some well-known lexical scoring functions are TF-IDF Robertson and Zaragoza (2009), Okapi BM25 Robertson and Zaragoza (2009), and SPLADE Formal et al. (2021). Specifically, each score computed for a term $w \in \mathcal{V}$ in the document D is a function of the frequency of w in D. To facilitate this, each document is modeled as a vector v over $\mathcal{V}$ whose entry for a term is nonzero only if the document contains that $\mathcal{V}$ term (hence *sparse*, since a document uses only a small fraction of the vocabulary). Scoring a *single* D against Q thus amounts to looking up v_D's entries at the coordinates named by the query's terms and summing them, i.e., a small set of index lookups on vector v_D, where the indices are exactly the sensitive input/query. The following exposition describes techniques without privacy in mind (thus, the indices are visible to the server).

Inverted Indexes. An end-to-end PPRAG query under sparse retrieval requires `srv` to score Q against *every* $D \in \mathcal{C}$ and return only the k highest-lexical-scoring documents. Computing the lexical score for every document is wasteful since many documents typically won't contain any terms in the query and will always score 0. A common way to filter out these documents is to organize documents using an abstract data type called an *inverted index*. Inverted indexes are dictionaries that map each

term $w \in \mathcal{V}$ to the set of documents in which w occurs, called the *posting list*. Retrieval protocols built on an inverted index avoid this exhaustive scan: `srv` uses the inverted index to fetch the posting list for each $w \in Q$ and scores Q against *only the relevant documents* that reside in the posting lists. We discuss two methods for instantiating an inverted index in this context: bin-based and block-based.

Bin-based BM25. An inverted index can be instantiated as a *hash table*, where each hash table bin stores the posting list for each hashed term. We refer to this simple approach as *bin-based BM25*. At query time, `srv` hashes each $w \in Q$ to a bin, fetches the posting lists stored there, scores the documents they contain, and returns the k highest-scoring ones to `clnt`. The **challenges** of applying PIR on top of a bin-based BM25 are: ① In non-privacy-preserving bin-based BM25, the `clnt` receives the top-k documents that `srv` locally filtered across posting lists; however, in a PPRAG rendition, the server does not see Q, thus, it cannot perform this task. ② In non-privacy-preserving bin-based BM25, to minimize collisions we need a large hash table; however, PIR computation complexity scales superlinearly with the size of the hash table. This introduces a trade-off: a large hash table yields fewer posting lists (i.e., less filtering) but increases PIR complexity, while a small hash table yields more posting lists (i.e., more filtering) with faster PIR complexity.

Block-based BM25 Ding and Suel (2011). In a typical RAG deployment, `clnt` sends Q to `srv` and receives only the k most similar documents in $\mathcal{C}$ w.r.t. Q. To speed up this search, `srv` can *efficiently dismiss* documents that cannot reach the top-k. We now discuss one such strategy built on the inverted index, *Dynamic Pruning* Turtle and Flood (1995); Broder et al. (2003). The underlying intuition is as follows. Recall that a document's D score is the sum of D's per-term contributions over the query terms, so for a "batch" $\mathcal{B} \subseteq \mathcal{C}$ of documents, taking the single largest contribution of each $w \in Q$ *across all* documents in $\mathcal{B}$ and summing[2] these maxima provides an upper bound for the exact score of *every* document in $\mathcal{B}$. If this over-approximation falls below the current top-k exact score, the entire batch can be skipped without scoring any of its documents individually. Dynamic Pruning realizes this idea as follows: At initialization, `srv` partitions all documents into *blocks* where each one contains a fixed number of documents. For each block $\mathcal{B}$ and term w, `srv` precomputes $\sigma_w(\mathcal{B})$, the largest score between w and any document in $\mathcal{B}$ (thus, $|\mathcal{V}|$ scores per block). At query time, `srv` retrieves the blocks for each $w \in Q$ and scores the first k documents arbitrarily to establish a baseline threshold. It then proceeds block by block: since no document in $\mathcal{B}$ can score higher than $\sum_{w \in Q} \sigma_w(\mathcal{B})$, a bound below the kth best exact score seen so far rules out the whole block, which is skipped, unscored. Blocks clearing the threshold are scored in full and the running top-k updated. Once all blocks are traversed, `srv` returns the final top-k; we refer to this as *Block-based BM25*.

Applying PIR to this procedure is challenging: ① The algorithm attempts to find the top scoring documents but, given the lack of organization among blocks, the server needs to linearly scan all blocks to identify the right ones. ② In non-private Block-based BM25, `srv` tracks the kth best score seen so far, a threshold that is *query-dependent*. Under PPRAG, `srv` never sees (sensitive) Q and therefore *cannot compute* this threshold, let alone decide which blocks it prunes.

3.2 Dense Retrieval and Privacy.

Whereas sparse retrieval matches on exact term overlap, dense retrieval matches on *meaning*, so a document can rank highly without sharing a single word with the query. Dense retrieval maps documents and queries into a shared vector space in which semantically similar texts lie close together Karpukhin et al. (2020); Murphy (2013). A popular measure of closeness is cosine similarity: given a query embedding $\mathbf{e}_Q \in \mathbb{R}^d$ and a document embedding $\mathbf{e}_D \in \mathbb{R}^d$, we compute $\cos(\mathbf{e}_Q, \mathbf{e}_D) = (\mathbf{e}_Q \cdot \mathbf{e}_D)/(\|\mathbf{e}_Q\|\|\mathbf{e}_D\|)$.

Approximate Nearest Neighbor Search. In a typical dense RAG deployment, `clnt` sends its query embedding $\mathbf{e}_Q$ to `srv`, which computes the cosine similarity against every document embedding and returns the k closest. This is exactly k-Nearest Neighbor (NN) search Har-Peled et al. (1998), at a cost of $O(|\mathcal{C}|d)$ per query. The common alternative is *Approximate* Nearest Neighbor (ANN) search Karpukhin et al. (2020); Zhou et al. (2024b), which in one popular form Zhou et al. (2024b)

[2]Note that no document in $\mathcal{B}$ need actually attain this bound, since the per-term maxima may come from different documents, e.g., D_3 for q_1, say, and D_9 for q_2.

represents $\mathcal{C}$ as a graph[3] whose nodes are documents and whose edges join embeddings of high cosine similarity, reducing retrieval to a traversal toward increasingly similar documents rather than a full scan. Dense retrieval poses its own challenges under PIR: ① The graph traversal is *query-dependent*: from an arbitrary starting node, `srv` scores $\mathbf{e}_Q$ against the current node's neighbors and moves to the closest one, so which node is accessed at each step is determined by the query. Revealing this access pattern would *leak information* about $\mathbf{e}_Q$. ② The number of traversal steps also varies with the query, since the search halts once it stops finding closer neighbors. This count is itself a function of $\mathbf{e}_Q$, so revealing it would *leak information* about the query. A PPRAG-friendly ANN must therefore fix the step count in advance, trading retrieval accuracy against cost on every query.

Properties of a Hybrid Retrieval Protocol. Sparse and dense retrieval succeed on complementary dimensions. Sparse retrieval pinpoints domain-specific terms that embeddings often fail to represent, while dense retrieval captures semantic relationships that exact lexical matching misses. A hybrid PIR protocol should therefore combine both, inheriting Ⓐ the low, *fixed round complexity* of sparse search, Ⓑ sparse retrieval's ability to *quickly dismiss documents* with no lexical overlap with the query, and Ⓒ the *semantic accuracy* of dense embeddings.

4 PILLAR: *PIR-Friendly* HYBRID RETRIEVAL

This section presents our protocols `PILLAR-Bin` and `PILLAR-Tree`, both of which realize PPRAG. Their design follows directly from the three properties a hybrid architecture should have, stated at the end of Section 3. Regarding Ⓐ, every adaptive component is removed. The number of rounds and the size of every message depend only on publicly known parameters, never on Q, so the communication pattern is fixed. For Ⓑ, the crude part of sparse retrieval runs first on the server, and the refined filtering stage on the client. Lexical scoring dismisses the bulk of $\mathcal{C}$ cheaply, leaving a small candidate set for the expensive stage. Ⓒ Semantic re-ranking is moved completely to `clnt`. After the sparse stage, `clnt` fetches the candidates' embeddings via PIR and ranks them locally, recovering the accuracy of dense retrieval without `srv` ever learning Q. We provide the full protocols for both `PILLAR-Bin` & `PILLAR-Tree` in Appendix C and the security analysis in Appendix E.

4.1 A SIMPLE BIN-BASED BM25

We discuss solutions for the two challenges identified for Bin-based BM25 in Section 3.1.

Solving Bin-Based BM25 Challenges. ① Recall the first problem for Bin-Based BM25 was that, in non-privacy-preserving Bin-Based BM25, `srv` would retrieve posting lists for each term $w \in Q$ and score the documents in the posting lists, returning just the top-k highest-scoring documents. Scoring documents requires access to Q, which `srv` does not see in a PPRAG protocol. To solve this, `srv` will only return a fixed number of t documents per query term to `clnt`, who can locally re-rank without revealing Q. To ensure the t returned documents are still relevant, we return documents from a posting list that is sorted by frequency of the hashed term $w \in \mathcal{V}$ used to retrieve the posting list, and the t highest-frequency documents are returned. Empirical approaches for choosing t are explored in Section 5. ② Recall the trade-off: a large hash table keeps collisions rare but drives up PIR complexity, while a small one requires less PIR operations but conflates unrelated terms into the same bin. `PILLAR-Bin` uses only $R \ll |\mathcal{C}|$ rows and maps each term w to row $H(w) \pmod R$ under a public hash function H. Thus, `clnt` can compute which row to request, and PIR hides that index from `srv`. R is also smaller than the vocabulary, so many terms share a row, and the postings lists of all colliding terms are merged into it. Rows are truncated at t length, so, crucially, the order of merging lists determines which documents survive truncation. Naively appending lists would preserve the first list in full and cut off the last list entirely. `PILLAR-Bin` therefore *interleaves* lists in a round-robin fashion, placing the jth document of the ith list at position $(j-1)m + i$ when m terms collide. Reading the first tm positions then yields the top t documents from each colliding list. Finally, the `srv` omits duplicates when a document is contained in two colliding lists.

[3]This converts a high-dimensional geometric search into a combinatorial one, with the geometry encoded into the edge set rather than evaluated at query time.

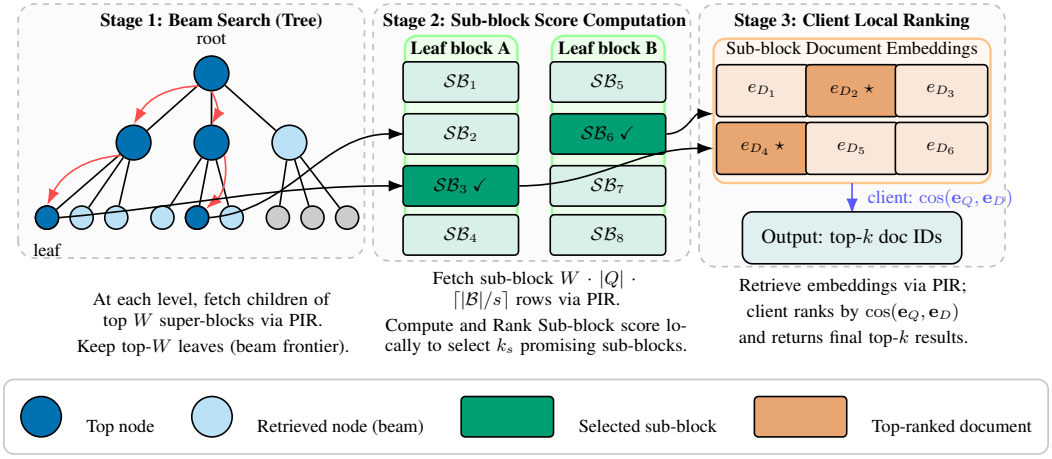


Figure 2: Three-stage PILLAR-Tree protocol illustration. The interaction are all private under PIR.

Final Design Description. At setup, srv computes the embedding $\mathbf{e}_D \in \mathbb{R}^d$ of every $D \in \mathcal{C}$ and stores embeddings in place of documents, so that clnt can re-rank locally once retrieval completes, i.e., the dense retrieval part of the hybrid design. It then builds the hash table as in non-private bin-based BM25 and applies the two designs above, i.e., hashing over R rows and interleaving colliding postings lists round-robin. The result is a PIR database of R rows, each holding exactly t embeddings (after truncation) and hence $t \cdot d$ field elements[4]. Since a query contains several terms, retrieving one row per term with independent PIR queries would cost $|Q|$ separate protocol executions; we instead use *batched PIR*, which packs multiple indices into a single query at sublinear amortized cost per index Zhou et al. (2024b). The batch size is fixed to the padding bound of ③ rather than to $|Q|$, so the query length is identical across executions and leaks nothing about how many terms the client used. clnt issues one batched PIR query, decodes the t embeddings in each returned row, and aggregates them into a candidate pool $\mathcal{D}_{\text{cand}}$ of at most t times the batch size, discarding duplicates that arise when a document appears in several retrieved rows. The entire protocol is a *single round*, and its message sizes depend on R, t, d, and the batch bound.

4.2 Tree-based BM25 (PILLAR-Tree)

This section discusses how to solve the previously identified challenges for block-based BM25 in Section 3.1 and provides the design principle for PILLAR-Tree.

Solving Block-Based BM25 Challenges. To resolve ①, we organize the blocks recursively into a tree whose leaves are the individual blocks and whose internal nodes each represent a *super-block* spanning the documents of all blocks in their subtree. At every node we store an array of $|\mathcal{V}|$ entries, where the i-th entry holds the maximum lexical score of term i over all documents in that super-block. For full generality, we consider r-ary trees, i.e., branching factor r, so as to reduce the height of the tree. Next, we descend the tree with a *beam search* Bisiani (1987) of width W, confine the next level traversal to the children of the highest scoring W nodes. Since the previous level advances only W super-blocks (and each super-block has r children), layer i considers only $W \times r$ candidate super-blocks. For each candidate, the algorithm sums the upper bounds of the query terms in Q and retains the W highest-scoring super-blocks of the level. It is worth noting that given that beam search is a heuristic technique, it doesn't guarantee to find the true top-k blocks (but our experiments show better-than-SOTA accuracy). Overall, the traversal descends $\lceil \log_r(|\mathcal{C}|/|\mathcal{B}|) \rceil$ levels and examines at most $W \cdot r$ nodes per level, for a total of $O\left(W \cdot r \cdot |Q| \cdot \log_r \frac{|\mathcal{C}|}{|\mathcal{B}|}\right)$ upper-bound array fetches. The baseline, in contrast, scans blocks linearly and requires $|Q| \frac{|\mathcal{C}|}{|\mathcal{B}|}$ array fetches, i.e., from linear to logarithmic. Regarding ②, the clnt proceeds interactively: at each level clnt retrieves via PIR the relevant array entries from the candidate super-blocks that clnt already identified, sums them locally to obtain the total upper bound for each candidate super-block, and uses the result to

[4]Thus, all rows are indistinguishable in size regardless of how many terms collide in them.

select which super-blocks become candidates at the next level. Finally, the bottom level stores not the embeddings themselves but the identifiers of the embeddings contained in each block. Retrieving the embeddings therefore requires one additional round of PIR, after which the client performs the dense retrieval step locally. Overall, the `clnt` performs $\log_r \frac{|\mathcal{C}|}{|\mathcal{B}|}$ PIR rounds, and at each round it batches $W \cdot r \cdot |Q|$ array indices to retrieve (here, we concatenate all arrays-of-upper-bounds of the level into a single one). We apply four further optimizations to improve accuracy and performance. First, at preprocessing time `srv` clusters the corpus so that each block contains mutually similar documents. Second, each PIR entry at a given node $\mathcal{B}$ and word w stores the scores of w w.r.t. r children of $\mathcal{B}$ rather than to $\mathcal{B}$ itself. At query time, this reduces the PIR cost from $|Q|Wr$ queries over a $|\mathcal{V}| \times r^{\ell+1}$-sized child-level PIR database to $|Q|W$ queries over a $|\mathcal{V}| \times r^{\ell}$-sized current-level PIR database. Third, we add a refinement layer below each leaf block, separate from the tree traversal. Each leaf contains $|\mathcal{B}|$ documents and is partitioned into sub-blocks of s documents, giving branch factor $r^\star = \lceil |\mathcal{B}|/s \rceil$. After the tree traversal filters the search space to $W \times |\mathcal{B}|$ candidate documents, the refinement layer selects the top k_s sub-blocks, reducing the embedding-fetch cost from $|\mathcal{B}| \times W$ embeddings to $k_s \times s$. Finally, to avoid storing $|\mathcal{V}|$ mostly-zero upper bounds per super-block at each level, we realize each level's PIR database as a cuckoo hash table holding only non-zero entries, trading m PIR queries per lookup for a substantial reduction in server storage and PIR cost (see Appendix F). There are many parameters that affect search speed and quality, such as branching factor, r, beam width, W, block size, $|\mathcal{B}|$, etc. we study the effect and the interplay of the different parameters in Appendix B.1. A high-level design of `PILLAR-Tree` is shown in Figure 2, where we split the design of `PILLAR-Tree` into three stages: In Stage 1, the `clnt` traverses the tree with beam search, using one PIR round per level, until it reaches W leaf blocks. In Stage 2, the `clnt` retrieves via PIR the upper bounds of the sub-blocks of these leaf blocks, selects the top k_s sub-blocks, and obtains the identifiers of their embeddings. In Stage 3, the `clnt` retrieves these embeddings via PIR and performs the dense retrieval locally.

5 Evaluation

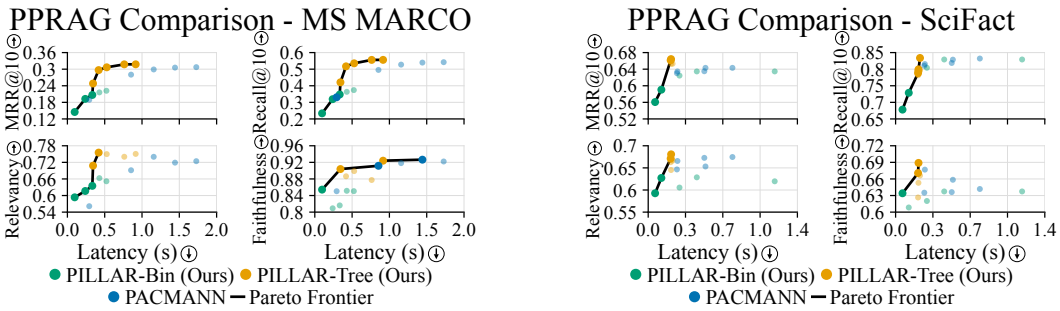
5.1 Experimental Setup

We implement our designs `PILLAR-Bin` and `PILLAR-Tree` (Section 4) in Go and compare them against PACMANN (Zhou et al., 2024b), the state-of-the-art PPRAG protocol, which answers each query through a multi-round private ANN traversal. For all three protocols, we run a hyperparameter grid search over all parameters (we detail the tunable parameters in Table 1). We plot/detail the metrics for every configuration of every method over our chosen datasets in Appendix B. Because the number of configurations is large, this section reports five representative configurations per protocol; Appendix B repeats each plot with all configurations included.

Param.	Meaning
	`PILLAR-Bin`
R	Hash table size
t	# Documents per bin
DB_{emb}	Retrieve embeddings directly or use two PIR rounds
	`PILLAR-Tree`
r	Branching factor of the tree
$\lvert\mathcal{B}\rvert$	# Documents per leaf block
s	# Documents per sub-block
W	Beam width
	PACMANN
p	# Steps in ANN traversal
n	# Neighbors retrieved per step

Table 1: The parameter grid searched for each protocol. Every combination was run on both SciFact and MS MARCO.

On Selecting Representative Configurations. We adapt a standard algorithm for determining how to minimize the metric *cost* while maximizing the metric *benefit* over multiple configurations, called *Kneedle* Satopaa et al. (2011). We define the benefit of a configuration by the mean of its MRR@10 and Recall@10, each min-max normalized first, which are standard metrics Thakur et al. (2021a) in determining the quality of documents retrieved. The ideal cost for a PPRAG must consider two quantities, (i) the number of PIR rounds it needs, and (ii) the total data transferred. We use per-query WAN latency as a proxy for both: each sequential PIR round adds a full network round trip, and each byte transferred adds transmission time over the limited bandwidth. Following Kneedle, for each of the three protocols, we compute and select five configurations from the Pareto frontier: The lowest-cost configuration (C1), the configuration with



(a) PPRAG Comparison - MS MARCO

(b) PPRAG Comparison - SciFact

Figure 3: Retrieval quality (MRR@10, Recall@10) and answer quality (Answer Relevancy, Faithfulness) (Y-axis, higher is better) against per-query WAN latency (X-axis, lower is better) for the five selected configurations of each protocol. The black line is the Pareto frontier, which highlights the best configurations. Configurations not on the Pareto frontier are slightly transparent.

the largest normalized benefit relative to its normalized cost (C2), the configurations with the highest MRR@10 (C3) and highest Recall@10 (C4), and the highest-cost configuration (C5). Appendix B details the full procedure. For completeness, we consider other instantiations of the metric cost, such as LAN latency, computation time, and total data sent, in Appendix B.

Datasets. We evaluate on the MS MARCO passage retrieval benchmark (Bajaj et al., 2018), consisting of 8.8 million passages and 6,980 queries, and on SciFact Wadden et al. (2020), a corpus of 5,183 documents and 300 queries. These two datasets cover the qualities that matter for a PPRAG evaluation: MS MARCO is large, so latency, communication, and memory all matter, and its queries are open-domain, so exact term overlap is a weak measure of similarity, which works against the lexical scoring components in both `PILLAR-Bin` and `PILLAR-Tree`. SciFact is three orders of magnitude smaller, so cost differences matter less, but its queries are scientific claims with rare, specific terminology, where lexical scoring is strongest.

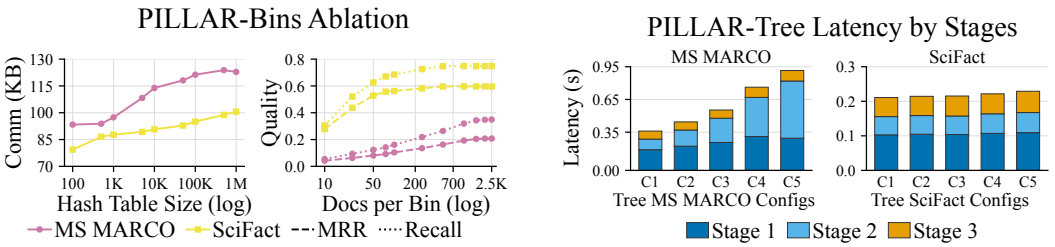
Testbed. Retrieval experiments were run on a Debian 12 container with a 64-core, 128-thread AMD EPYC Zen 2 CPU, but we limit each process to only 15 threads. For end-to-end answer quality, the documents retrieved by each selected configuration are passed to an LLM, Qwen2.5-7B-Instruct Qwen et al. (2025). The resulting answers are scored by the RAGAS framework Es et al. (2025), with Llama-3-70B Grattafiori et al. (2024) as judge. To simulate WAN latency, we use a round-trip-time of 50 milliseconds and 400 megabits per second bandwidth. See further details in G.1

5.2 Comparison against PACMANN and Ablation

Metrics. To benchmark our proposed PPRAGs, we use Recall@10, which measures whether the correct document appears among the top 10 retrieved, and MRR@10, which rewards placing the correct document near the top. We also use RAGAS metrics, *Faithfulness* measures whether the claims in the LLM's answer are supported by the retrieved documents, and *Answer Relevancy* measures whether the LLM actually answered the question, regardless of the presence of the correct document. We report both groups, because MRR and Recall alone do not show whether the retrieved context is useful to an LLM[5]. As all of our configurations offer a cost-benefit trade-off, we show the *Pareto frontier* (black line) in Figure 3 for each configuration in our two datasets. If a configuration lies on the Pareto frontier, no configuration of any protocol achieves better quality (Y-axis) at equal or lower latency (X-axis). Across all of our experiments (See Figure 3), 93% of the points on the Pareto frontier come from our proposed PILLAR hybrid architecture and only 7% come from PACMANN.

On the Analysis of The Pareto Frontier. `PILLAR-Bin` retrieves the top 10 documents in as little as 0.098 s on MS MARCO and 0.065 s on SciFact. There is no configuration of PACMANN (on the Pareto frontier) that runs under 0.1 s on either datasets (while `PILLAR-Bin` has multiple), the fastest PACMANN configuration takes 0.293 s on MS MARCO and 0.270 s on SciFact, which

[5]The evaluation of PACMANN only considers MRR and Recall, but a RAG pipeline needs to retrieve documents that generate a well-informed answer, so we report RAGAS metrics alongside MRR and Recall



(a) `PILLAR-Bin` Ablation

(b) `PILLAR-Tree` Latency by Stages

Figure 4: Left: Two `PILLAR-Bin` ablation plots, total communication cost per query against the size of the hash table (log scale), and retrieval quality against documents stored in a bin (log scale). Right: per-query latency of the five selected `PILLAR-Tree` configurations (C1-C5) (defined in Section 5.1 and Table 3 lists their specific parameters) by the 3 stages defined in Section 4.2.

is $3\times$ to $4\times$ slower. As for `PILLAR-Tree`, in terms of speed, `PILLAR-Tree` rarely reaches `PILLAR-Bin`'s latencies, since its traversal needs $\lceil \log_r(|\mathcal{C}|/|\mathcal{B}|) \rceil$ rounds. In terms of quality, across all configurations on the Pareto frontier in Figure 3 that run in under 1.0 s, `PILLAR-Tree` achieves the highest quality across all metrics and across both datasets. Across all configurations used to produce Figure 3, 54% of the points are configurations of `PILLAR-Tree`.

`PILLAR-Tree` Latency by Stages. Many `PILLAR-Tree` configurations retrieve documents that score highly across quality metrics, but, for some of them, the latency is increased. To investigate this, we show how each of the three stages of `PILLAR-Tree` affects latency in Figure 4b. On MS MARCO, Stages 1 and 3 stay roughly constant, which are the traversal and embedding retrieval stages, respectively. Almost all latency variability comes from Stage 2, the sub-block retrieval stage. Stage 2 latency drives retrieval quality; the highest-latency configuration (C5) reaches 0.318 MRR on MS MARCO, while the lowest-latency configuration reaches 0.248, a 28% gain. W determines how many candidate documents Stage 2 refines, which makes it the first parameter to tune on a new dataset. On SciFact, the five configurations differ by under 20 ms. This is due to the size of the tree, which will always be smaller on a small dataset (such as SciFact).

`PILLAR-Bin` Ablation. We test the two parameters of `PILLAR-Bin`: the size of the hash table, and the number of documents stored in a bin. Specifically, we investigate (i) how the size of the hash table affects the communication cost, as well as (ii) how the number of documents stored in a bin affects retrieval quality (we provide a more extensive ablation on `PILLAR-Bin` in Appendix B). The left plot of Figure 4a fixes the documents per bin at 10 and varies the hash table size (X-axis), showing that total communication cost per query (Y-axis, Comm KB) increases with the size of the hash table (as is expected when using PIR). We vary the size of the hash table by 6 orders of magnitude, but see that the communication cost for the largest hash table is not even $2\times$ the cost of the smallest. Thus, hash table size has little impact on the total communication cost, allowing flexibility to choose the hash table size that provides the best retrieval quality (as shown in Appendix B). The right plot of Figure 4a fixes the hash table sizes at 1 million for MS MARCO and 5 thousand for SciFact and varies the number of documents stored in a bin (X-axis). We see that MRR@10/Recall@10 (Y-axis, Quality) for the retrieved documents increases as the number of documents stored in a bin increases. Scifact sees diminishing returns, after storing 700 documents per bin, there is little more retrieval quality to gain. MS Marco continues to see benefits from more returned documents, even at the highest tested values, 1.0K-2.5K documents per bin, MS MARCO still increases.

6 Conclusion

We presented PILLAR, a pair of PIR-friendly hybrid retrieval protocols (`PILLAR-Bin` and `PILLAR-Tree`) that split retrieval between a cheap lexical score and a semantic re-ranking performed entirely by the client. We evaluate our two protocols using the state-of-the-art, PACMANN. `PILLAR-Bin` always has a faster configuration available, answering queries three times faster than any configuration PACMANN admits, while `PILLAR-Tree` frequently retrieves better quality documents than PACMANN at the same speed.

AI USE STATEMENT

In this work, we used generative AI tools to assist in implementing methods, namely writing code for our protocols and result plots. We have not used generative AI tools to generate synthetic datasets, 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, clean or reformat datasets, or interpret results, and assistance with translation and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to draft parts of the paper (initial text, section structure and titles, and LaTeX layout), to edit the paper for grammar, spelling, and readability, and to identify and summarize relevant literature. We have reviewed all AI-assisted work. All AI-generated code was reviewed, re-written, tested, and validated for correctness by the authors. AI-drafted text served only as a starting point and was revised by the authors, and all technical content, claims, and proofs are the authors' own. Literature identified or summarized with AI assistance was checked against the original sources, and all citations were verified manually. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY

We provide an extensive Appendix that details the step-by-step operation of `PILLAR-Tree` and `PILLAR-Bin` (Appendix C) and we also detail all the different parameters tested for all three protocols (Appendix B and Table 1). We have a codebase entirely in Go, that can be used to reproduce our experiments. Upon full release we will make our codebase public.

REFERENCES

Ishtiyaque Ahmad, Laboni Sarker, Divyakant Agrawal, Amr El Abbadi, and Trinabh Gupta. Coeus: A system for oblivious document ranking and retrieval. In *Proceedings of the 28th ACM Symposium on Operating Systems Principles (SOSP)*, pages 672–690, 2021.

Anthropic. Introducing contextual retrieval. `https://www.anthropic.com/news/contextual-retrieval`, 2024. Accessed September 2026.

Martin Aumüller, Erik Bernhardsson, and Alexander John Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *CoRR*, abs/1807.05614, 2018. URL `http://arxiv.org/abs/1807.05614`.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. Ms marco: A human generated machine reading comprehension dataset, 2018. URL `https://arxiv.org/abs/1611.09268`.

Michael Barbaro and Tom Zeller Jr. A face is exposed for AOL searcher no. 4417749. The New York Times, 2006. URL `https://archive.nytimes.com/www.nytimes.com/learning/teachers/featured_articles/20060810thursday.html`. Accessed September 2026.

Amos Beimel, Yuval Ishai, and Tal Malkin. Reducing the servers computation in private information retrieval: Pir with preprocessing. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, pages 55–73, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-44598-2.

Alec Berntson, Alina Stoica Beck, Amaia Salvador Aguilera, Farzad Sunavala, Thibault Gisselbrecht, and Xianshun Chen. Raising the bar for rag excellence. Whitepaper, `https://cdn-dynmedia-1.microsoft.com/is/content/microsoftcorp/microsoft/final/en-us/microsoft-brand/documents/ai-search-query-engine-performance-whitepaper.pdf`. Accessed September 2026.

Roberto Bisiani. Beam search. In Stuart C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*. Wiley, 1987.

Andrei Broder, David Carmel, Michael Herscovici, Aya Soffer, and Jason Zien. Efficient query evaluation using a two-level retrieval process. In *CIKM*, 2003.

Sebastian Bruch, Siyu Gai, and Amir Ingber. An analysis of fusion functions for hybrid retrieval. *ACM Transactions on Information Systems*, 2023.

Benny Chor and Niv Gilboa. Computationally private information retrieval (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '97, page 304–313, New York, NY, USA, 1997. Association for Computing Machinery. ISBN 0897918886. doi: 10.1145/258533.258609. URL https://doi.org/10.1145/258533.258609.

Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, 1995.

Gordon V. Cormack, Charles L A Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, page 758–759, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605584836. doi: 10.1145/1571941.1572114. URL https://doi.org/10.1145/1571941.1572114.

Henry Corrigan-Gibbs and Dmitry Kogan. Private information retrieval with sublinear online time. In *Advances in Cryptology – EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I*, page 44–75, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-45720-4. doi: 10.1007/978-3-030-45721-1_3. URL https://doi.org/10.1007/978-3-030-45721-1_3.

Shuai Ding and Torsten Suel. Faster top-k document retrieval using block-max indexes. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, page 993–1002, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450307574. doi: 10.1145/2009916.2010048. URL https://doi.org/10.1145/2009916.2010048.

Elastic. Reciprocal rank fusion. Elasticsearch Reference, https://www.elastic.co/docs/reference/elasticsearch/rest-apis/reciprocal-rank-fusion. Accessed September 2026.

Shahul Es, Jithin James, Luis Espinosa-Anke, and Steven Schockaert. Ragas: Automated evaluation of retrieval augmented generation, 2025. URL https://arxiv.org/abs/2309.15217.

Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. SPLADE: sparse lexical and expansion model for first stage ranking. *CoRR*, abs/2107.05720, 2021. URL https://arxiv.org/abs/2107.05720.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*, 2023.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, Danny Wyatt, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Francisco Guzmán, Frank Zhang, Gabriel

Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Govind Thattai, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jack Zhang, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Karthik Prasad, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Kushal Lakhotia, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Maria Tsimpoukelli, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Ning Zhang, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohan Maheswari, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vítor Albiero, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaofang Wang, Xiaoqing Ellen Tan, Xide Xia, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aayushi Srivastava, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Amos Teo, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Dong, Annie Franco, Anuj Goyal, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Ce Liu, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Cynthia Gao, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Eric-Tuan Le, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Filippos Kokkinos, Firat Ozgenel, Francesco Caggioni, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hakan Inan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Hongyuan Zhan, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Ilias Leontiadis, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Janice Lam, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kiran Jagadeesh, Kun Huang, Kunal Chawla, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Miao Liu, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang,

Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikhil Mehta, Nikolay Pavlovich Laptev, Ning Dong, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Rangaprabhu Parthasarathy, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Russ Howes, Ruty Rinott, Sachin Mehta, Sachin Siby, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Mahajan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shishir Patil, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Summer Deng, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Koehler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaojian Wu, Xiaolan Wang, Xilun Wu, Xinbo Gao, Yaniv Kleinman, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yu Zhao, Yuchen Hao, Yundi Qian, Yunlu Li, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, Zhiwei Zhao, and Zhiyu Ma. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. *Theory of Computing*, 8:321–350, 01 1998. doi: 10.4086/toc.2012.v008a014.

Alexandra Henzinger, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. Private web search with tiptoe. Cryptology ePrint Archive, Paper 2023/1438, 2023. URL https://eprint.iacr.org/2023/1438.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. In *IEEE Transactions on Big Data*, 2019.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.550. URL https://aclanthology.org/2020.emnlp-main.550/.

E. Kushilevitz and R. Ostrovsky. Replication is not needed: single database, computationally-private information retrieval. In *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, FOCS '97, page 364, USA, 1997. IEEE Computer Society. ISBN 0818681977.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

Xueguang Ma, Kai Sun, Ronak Pradeep, and Jimmy Lin. A replication study of dense passage retriever. *arXiv preprint arXiv:2104.05740*, 2021.

Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE TPAMI*, 2018.

Microsoft. Hybrid search using vectors and full text in Azure AI Search. https://learn.microsoft.com/en-us/azure/search/hybrid-search-overview. Accessed September 2026.

Kaisa Miettinen. *Nonlinear Multiobjective Optimization*, volume 12 of *International Series in Operations Research and Management Science*. Springer, 1999. doi: 10.1007/978-1-4615-5563-6.

Niloofar Mireshghallah, Maria Antoniak, Yash More, Yejin Choi, and Golnoosh Farnadi. Trust no bot: Discovering personal disclosures in human-LLM conversations in the wild. In *Conference on Language Modeling (COLM)*, 2024.

Kevin P. Murphy. *Machine learning : a probabilistic perspective*. MIT Press, Cambridge, Mass. [u.a.], 2013. ISBN 9780262018029 0262018020. URL https://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ref=sr_1_2?ie=UTF8&qid=1336857747&sr=8-2.

OpenSearch Project. Hybrid search. https://opensearch.org/docs/latest/search-plugins/hybrid-search. Accessed September 2026.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1410. URL https://aclanthology.org/D19-1410/.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends in IR*, 2009.

Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 166–171. IEEE, 2011. doi: 10.1109/ICDCSW.2011.20.

Robin Staab, Mark Vero, Mislav Balunović, and Martin Vechev. Beyond memorization: Violating privacy via inference with large language models. In *International Conference on Learning Representations (ICLR)*, 2024.

Nandan Thakur, Nils Reimers, Andreas Ruckl'e, Abhishek Srivastava, and Iryna Gurevych. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *ArXiv*, abs/2104.08663, 2021a. URL https://api.semanticscholar.org/CorpusID:233296016.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *NeurIPS Datasets and Benchmarks Track*, 2021b.

Howard Turtle and James Flood. Query evaluation: Strategies and optimizations. *Information Processing & Management*, 31(6):831–850, 1995. ISSN 0306-4573. doi: https://doi.org/10.1016/0306-4573(95)00020-H. URL https://www.sciencedirect.com/science/article/pii/030645739500020H.

David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. Fact or fiction: Verifying scientific claims. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7534–7550, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.609. URL https://aclanthology.org/2020.emnlp-main.609/.

Parameter	Definition
$\mathcal{C}$	The corpus
$\mathcal{V}$	Set of all unique terms in $\mathcal{C}$
d	Embedding vector dimension
D	Documents in $\mathcal{C}$
e_D	Embedding vector of document D
Q	Client query
e_Q	Embedding vector of client query
k	Number of relevant document retrieved
$\mathsf{id}(\cdot)$	Function to represent the index of a document/ document block
t	Number of documents returned per query term
R	Number of rows used in `PILLAR-Bin`
H	The public hash function used in `PILLAR-Bin`
B	Block Size
$\mathcal{B}$	Block of B documents in the corpus
$\sigma_w(\mathcal{B})$	Largest score between term w and any document in $\mathcal{B}$
r	Branching factor of the tree used in `PILLAR-Tree`
$\mathcal{H}$	Tree height
W	Beam width used in `PILLAR-Tree`'s beam search
s	Sub-block size
$q^\star$	Fixed query size, attained by pad/truncate the client query
k_s	Number of sub-block with document embeddings being retrieved

Table 2: Definition of variables used in the paper

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. Text embeddings by weakly-supervised contrastive pre-training, 2024. URL https://arxiv.org/abs/2212.03533.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, 2020. URL https://arxiv.org/abs/2002.10957.

Weaviate. Hybrid search. https://docs.weaviate.io/weaviate/concepts/search/hybrid-search. Accessed September 2026.

Mingxun Zhou, Andrew Park, Wenting Zheng, and Elaine Shi. Piano: Extremely simple, single-server pir with sublinear server computation. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4296–4314, 2024a. doi: 10.1109/SP54263.2024.00055.

Mingxun Zhou, Elaine Shi, and Giulia Fanti. Pacmann: Efficient private approximate nearest neighbor search. Cryptology ePrint Archive, Paper 2024/1600, 2024b. URL https://eprint.iacr.org/2024/1600.

Jinhao Zhu, Liana Patel, Matei Zaharia, and Raluca Ada Popa. Compass: Encrypted semantic search with high accuracy. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 915–938, 2025.

A Notation

In the paper we use different hyper-parameters and variables for our protocols. We detailed the parameters notations and their corresponding definition in Table 2.

B Additional Experiment Results

In this section we present additional result of all configurations that we use some of them in the main paper.

B.1 HYPERPARAMETER TUNING ON PILLAR-TREE CONFIGURATIONS

We perform a grid search on PILLAR-Tree hyperparameters and evaluate recall, MRR@10, and estimated number of PIR calls needed. From the evaluation results, we select one representative operating points from the Pareto frontier shown in Figure 5 using a defined process detailed later, together with the four configurations attaining the lowest PIR call, the highest PIR calls, the highest MRR@10 and highest Recall@10. We then run a full evaluation on these 5 configurations: we (1) evaluate the MRR@10 and Recall@10 of the configurations on a test set of $1,500$ queries from MSMARCO train set for MSMARCO, and a test set of 75 queries from Scifact train set for Scifact– to make sure tuning set and test set are different – (2) execute each configuration under PIR to benchmark runtime and (3) use RAGAS to evaluate downstream generation quality. For MSMarco, the search is performed over $B \in \{16, 32, 64\}, r \in \{8, 32, 128\}, q^\star \in \{2, 4, 6, 8\}, W \in \{50, 100, 200, 400\}, s \in \{4, 8, 16\}$. For Scifact, the search is performed over $B \in \{8, 16, 32, 64\}, r \in \{4, 8, 32, 128\}, q^\star \in \{4, 8, 12, 16\}, W \in \{5, 10, 20, 40, 80\}, s \in \{2, 4, 8\}$. The set of 5 candidate configurations along with their corresponding MRR are shown in Table 3.

B.1.1 REPRESENTATIVE CONFIGURATIONS SELECTION.

As discussed earlier, we use Pareto frontier to choose a representative configuration along with two configurations represent the best recall@10 and best MRR@10. Here we show how we choose the five configurations.

Pareto Frontier Let Θ denote the evaluated hyperparameter configurations. Following the standard definition of Pareto optimality Miettinen (1999), a configuration θ' dominates θ, denoted $\theta' \succ \theta$, if

$$\mathsf{MRR}(\theta') \geq \mathsf{MRR}(\theta), \qquad \mathsf{Recall}(\theta') \geq \mathsf{Recall}(\theta), \qquad \mathsf{PIRCall}(\theta') \leq \mathsf{PIRCall}(\theta),$$

with at least one strict inequality. The Pareto frontier is the set of configurations where improving one metric worsen at least another:

$$\mathcal{P} = \{\theta \in \Theta : \nexists \theta' \in \Theta \text{ such that } \theta' \succ \theta\}.$$

Effectiveness Score. To visualize the effectiveness-cost trade-off, we min-max normalize each retrieval metric:

$$\widetilde{\mathsf{MRR}}(\theta) = \frac{\mathsf{MRR}(\theta) - \mathsf{MRR}_{\min}}{\mathsf{MRR}_{\max} - \mathsf{MRR}_{\min}}, \qquad \widetilde{\mathsf{Recall}}(\theta) = \frac{\mathsf{Recall}(\theta) - \mathsf{Recall}_{\min}}{\mathsf{Recall}_{\max} - \mathsf{Recall}_{\min}}.$$

We define the aggregate effectiveness score as

$$E(\theta) = \frac{\widetilde{\mathsf{MRR}}(\theta) + \widetilde{\mathsf{Recall}}(\theta)}{2}.$$

The effectiveness-cost envelope $\mathcal{P}_E$ is the set when maximizing E and minimizing PIRCall:

$$\mathcal{P}_E = \Big\{\theta \in \Theta : \nexists\, \theta' \in \Theta \text{ such that } E(\theta') \geq E(\theta) \wedge \mathsf{PIRCall}(\theta') \leq \mathsf{PIRCall}(\theta), \\ \text{with at least one strict inequality}\Big\}.$$

Following the normalized difference construction of Kneedle Satopaa et al. (2011), we define

$$x(\theta) = \frac{\log \mathsf{PIRCall}(\theta) - \log \mathsf{PIRCall}_{\min}}{\log \mathsf{PIRCall}_{\max} - \log \mathsf{PIRCall}_{\min}}, \qquad y(\theta) = \frac{E(\theta) - E_{\min}}{E_{\max} - E_{\min}}.$$

The knee point, which represents the point on the envelope which witness the strongest gain in effectiveness with respect to its PIR cost, is defined as $\theta_{\text{knee}} = \arg\max_{\theta \in \mathcal{P}_E} (y(\theta) - x(\theta))$.

We then chooses the points C1, C2, C3, C4, C5 as:

- C1: Point C1 with lowest number of PIR calls
- C2: Point C2 is the knee point θ_{knee}

Dataset	Config	B	r	$q^\star$	W	s	k_s	Tuning		Test		PIR calls / query
								MRR@10	Recall@10	MRR@10	Recall@10	
MS MARCO	C1	64	128	2	50	16	32	0.1833	0.3055	0.1962	0.3426	**472**
	C2	64	128	4	50	16	64	0.2833	0.4880	0.3190	0.5751	944
	C3	16	128	8	400	8	256	**0.3218**	0.5639	0.3551	0.6573	13616
	C4	64	128	8	400	4	256	0.3211	**0.5649**	0.3549	**0.6631**	13216
	C5	16	8	8	400	16	256	0.3213	0.5625	**0.3556**	0.6580	28352
SciFact	C1	64	128	4	5	8	5	0.5303	0.5909	0.5852	0.6333	**53**
	C2	16	128	12	10	4	10	0.6623	0.8008	0.6192	0.7600	346
	C3	16	128	16	20	4	10	**0.6697**	0.8043	**0.6350**	0.7867	778
	C4	32	128	12	80	2	40	0.6606	**0.8428**	0.6225	0.8000	2032
	C5	8	4	16	80	4	80	0.6518	0.8267	0.6249	**0.8067**	6992

Table 3: Dense-only configurations selected from the plaintext parameter search. C1 has the lowest padded PIR cost, C2 is the effectiveness–cost knee, C3 has the highest tuning MRR@10, C4 has the highest tuning Recall@10, and C5 has the highest padded PIR cost. Tuning results use 6,980 MS MARCO queries and 300 SciFact queries; test results use 1,500 MS MARCO queries and 75 held-out SciFact queries. Bold metric values are column maxima within each dataset, while bold PIR costs are column minima. PIR-query counts assume constant-work padding and include two Cuckoo calls for every Stage 1 and Stage 2 lookup and one query for each retrieved embedding row. They count primitive PIR queries rather than network round trips after batching.

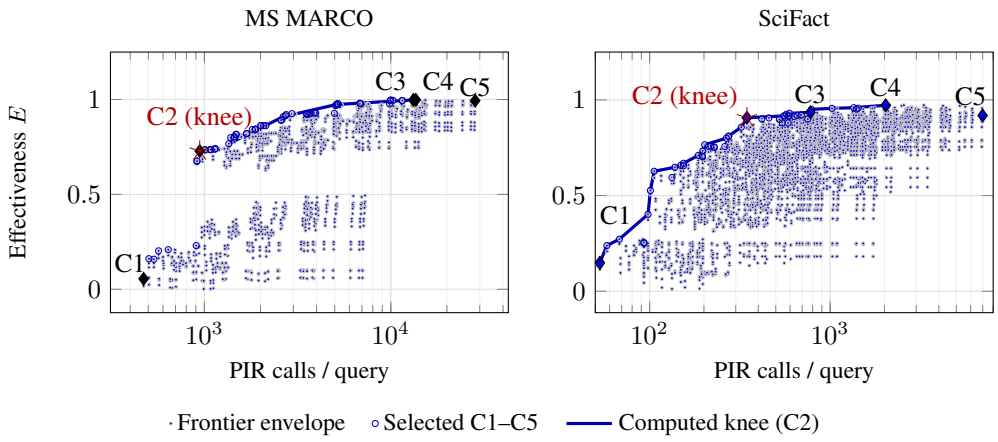


Figure 5: Effectiveness-PIR-cost trade-off over the complete plaintext grid. Blue marks are all tested configurations and circles are configurations nondominated in MRR@10, Recall@10, and padded PIR calls. The solid curve is the upper envelope $\mathcal{P}_E$, C1,C2,C3,C4,C5 are chosen configurations.

- C3: Point C3 with highest MRR@10 on tune set
- C4: Point C4 with highest Recall@10 on tune set
- C5: Point C5 with highest number of PIR calls

PACMANN and `PILLAR-Bin` follow a similar protocol to `PILLAR-Tree`, but instead they *directly* use the recorded WAN time as the cost. We detail PACMANN's best configs in Table 5 and `PILLAR-Bin`'s best in Table 4.

B.2 Extensive comparisons

In this section, we plot every different configuration we ran across PACMANN, `PILLAR-Bin` and `PILLAR-Tree`. We use the same Pareot frontiers as were in Section 5. We have over 2,000 total configurations, which is why we limited our comparison previously. To keep each plot legible, we only show the entire Pareto frontier and circled points on the frontier with a black outline. To get Faithfulness and Answer Relevancy for every single different configuration using the GPUs available

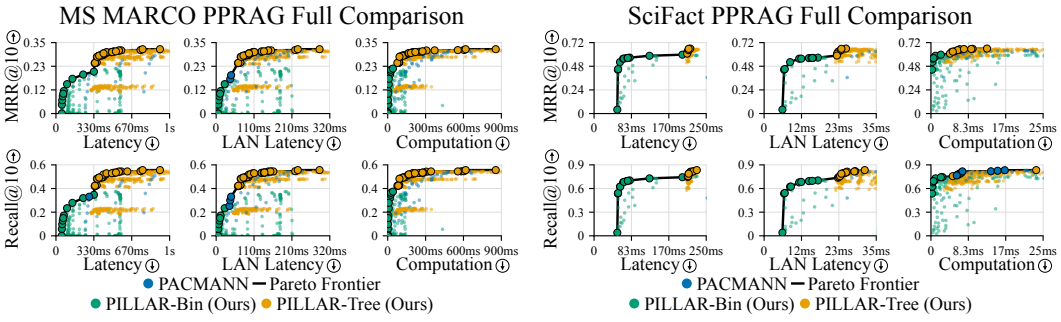
Dataset	Config	bs	dpb	MRR@10	Recall@10	WAN time / query (s)
MS MARCO	C1	10^6	250	0.145	0.2337	**0.098**
	C2	10^6	1500	0.2067	0.3474	0.334
	C3	10^6	2000	0.2160	0.3635	0.429
	C4	10^6	2500	**0.2219**	**0.3739**	0.523
	C5	10^6	1000	0.1925	0.3192	0.239
SciFact	C1	10^6	50	0.5607	0.6781	**0.065**
	C2	10^6	250	0.5905	0.7287	0.124
	C3	100	1500	0.6245	0.8042	0.294
	C4	100	2500	**0.6346**	**0.8292**	1.187
	C5	100	2500	**0.6346**	**0.8292**	0.456

Table 4: Bins configurations selected from the plaintext parameter search.

Dataset	Config	bs	dpb	MRR@10	Recall@10	WAN time / query (s)
MS MARCO	C1	5	48	0.1892	0.3306	**0.293**
	C2	15	40	0.2802	0.4955	0.853
	C3	20	48	0.2990	0.5275	1.157
	C4	30	48	**0.3073**	**0.5422**	1.727
	C5	25	48	0.3057	0.5397	1.442
SciFact	C1	5	40	0.6304	0.8104	**0.270**
	C2	5	48	0.6345	0.8154	0.275
	C3	10	40	**0.6432**	0.8287	0.537
	C4	15	32	**0.6432**	**0.8321**	0.791
	C5	10	32	0.6350	0.8187	0.527

Table 5: PACMANN configurations selected from the plaintext parameter search.

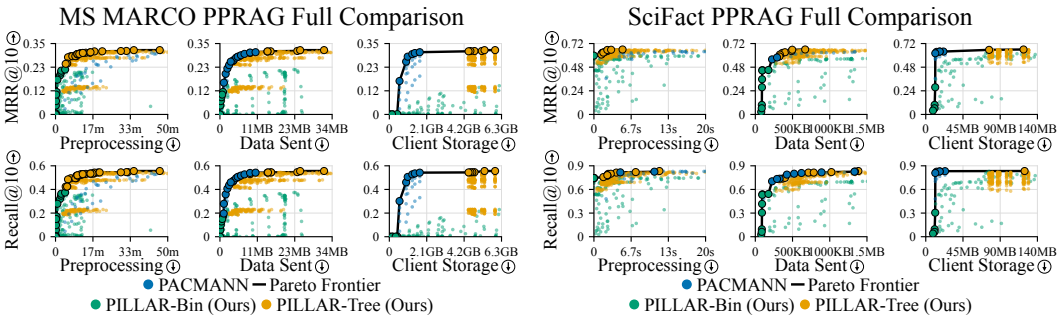
to us would've taken a substantial amount of time, as a result, we decided to include more total configurations but with more basic metrics. Every plot here only uses MRR and Recall



(a) MS MARCO (b) SciFact

Figure 6: MRR@10 and Recall@10 against per-query WAN latency, LAN latency, and computation time. Highlighted points lie on the global Pareto frontier across all three methods.

On the Time Costs. Figure 6 reports quality against per-query WAN latency, LAN latency, and computation time. If we fix the budget at the x-axis limits (1 s WAN, 320 ms LAN, and 900 ms computation on MS MARCO; 250 ms, 35 ms, and 25 ms on SciFact), `PILLAR-Tree` achieves the highest MRR@10 and Recall@10 in every panel. `PILLAR-Bin` occupies the low-latency end of every frontier. On SciFact, `PILLAR-Bin` already exceeds 0.5 MRR@10 within 83 ms of WAN latency and accounts for 50.0% of all frontier points. PACMANN has no configuration on the Pareto frontier in eight of the twelve panels, so it is never the Pareto-optimal choice. These eight panels are MRR@10 under WAN latency and both metrics under computation on MS MARCO,



(a) MS MARCO (b) SciFact

Figure 7: MRR@10 and Recall@10 against one-time PIR preprocessing time, per-query data sent, and one-time client storage. Highlighted points lie on the global Pareto frontier across all three methods.

and every panel on SciFact except Recall@10 under computation. In the remaining four panels, PACMANN contributes at most 5 points. Across all 377 frontier points, 97.1% are configurations of `PILLAR-Bin` or `PILLAR-Tree`.

On the Preprocessing, Communication, and Storage Costs. Figure 7 reports quality against PIR preprocessing time, data sent, and client storage. All costs are per query except PIR preprocessing and client storage, which are for the entire end-to-end run. *Preprocessing.* PACMANN has no configuration on the Pareto frontier on MS MARCO or for MRR@10 on SciFact. This is because its additional rounds of PIR queries require correspondingly more PIR preprocessing. At the x-axis limits (50 min on MS MARCO, 20 s on SciFact), `PILLAR-Tree` achieves the highest quality on both metrics. *Data sent.* At the x-axis limits (34 MB on MS MARCO, 1.5 MB on SciFact), `PILLAR-Tree` again achieves the highest quality. PACMANN occupies the middle of the frontier on MS MARCO (51.9% of points for MRR@10, 51.7% for Recall@10) but only 11.8% and 39.1% on SciFact. Even though PACMANN may often be the choice for a low amount of total communication, as we have shown previously, the many rounds practically drag down WAN and LAN latency. *Client storage.* PACMANN's many rounds of interaction let it keep the client-side cost low, and it reaches near-maximal quality with under 2.1 GB on MS MARCO and under 45 MB on SciFact, whereas `PILLAR-Tree`'s highest-quality configurations require over 4.2 GB and 90 MB, respectively. The cost of these rounds is paid in latency, as shown in Figure 6. Despite this, PACMANN does not hold a majority of the frontier, it only has 35.3% and 33.3% of the points on MS MARCO (tied with `PILLAR-Tree`), and 45.5% and 50.0% on SciFact. `PILLAR-Tree` still achieves the highest quality at the x-axis limits (6.3 GB and 140 MB). Moreover, on MS MARCO nearly every configuration that is viable of every method requires client storage on the order of gigabytes, so PACMANN's advantage does not change the scale of the client's cost. Across all 262 frontier points in Figure 7, 75.6% are configurations of `PILLAR-Bin` or `PILLAR-Tree`, with `PILLAR-Tree` the largest share on both datasets (48.0% and 41.6%).

B.3 Further `PILLAR-Bin` ablation plots

We extend the `PILLAR-Bin` ablation of Section 5 with system costs, client storage and preprocessing time. As in the main text, when varying the hash table size we fix the number of documents per bin at 10, and when varying the number of documents per bin we fix the hash table size at 1M for MS MARCO and 5K for SciFact. WAN latency, LAN latency, compute (total running time) and data sent (upload and download) are per query. Client storage and PIR preprocessing total costs over the entire run.

WAN, LAN and total compute. Figure 8 (left) shows that WAN latency, LAN latency and compute are either not affected by the hash table size increases or show no consistent trend. Only data sent increases, the same trade-off we discussed in Section 5 Figure 8 (right) shows that all four costs increase with the number of documents per bin. For SciFact, the increase is small up to 250 documents per bin and steep between 250 and 500.

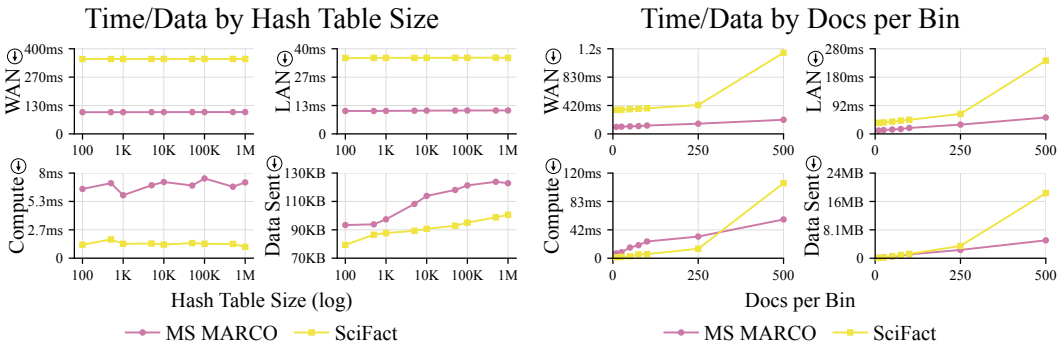


Figure 8: `PILLAR-Bin` per-query cost. Left: varying the hash table size. Right: varying the number of documents per bin.

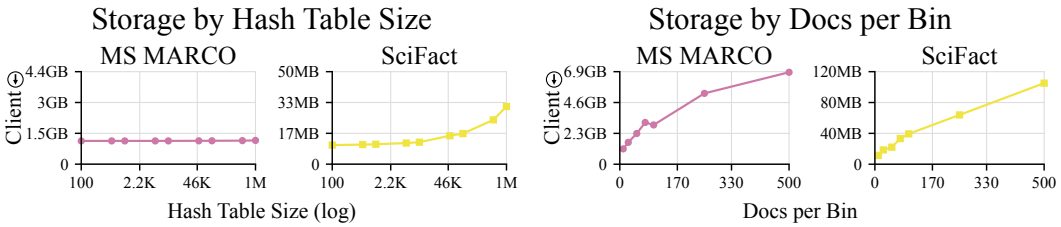


Figure 9: `PILLAR-Bin` worst-case client storage. Left: varying the hash table size. Right: varying the number of documents per bin.

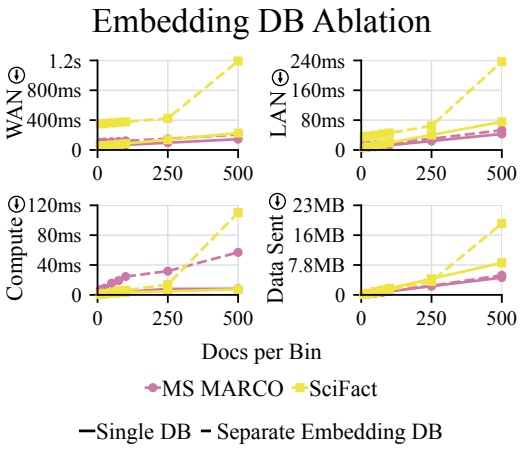


Figure 11: `PILLAR-Bin` per-query cost when embeddings are stored in the bins (Single DB) or in a separate database (Separate Embedding DB), varying the number of documents per bin.

Storage costs and PIR preprocessing. Figure 9 reports client storage in the worst case, where every hint is completely full. In practice, client storage is dependent on how many bins are full and ranges from 30-50% better. For MS MARCO, client storage is approximately 1GB and does not change with the hash table size. As we saw in the ablation comparing against PACMANN, `PILLAR-Bin` that take up a large client hint tend to not perform well regardless. For SciFact, client storage increases from 11MB at a hash table size of 100 to 30MB at 1M, with most of the increase above 10K. Unlike the hash table size, the number of documents per bin has a large effect on client storage, which increases for SciFact and for MS MARCO.Figure 10 (top) shows that, surprisingly, PIR preprocessing time does not depend on the hash table size. In our implementation, we made heavy use of multithreading to perform pre-processing, so it is likely that the bottleneck was thrashing related, as a large hash table size (should) imply more entries to retrieve PIR hints from. Preprocessing time increases with the number of documents per bin, from 0.6 minutes at 10 to 4.9 minutes at 500 for MS MARCO, and from a few seconds at up to 100 to 1.7 minutes at 500 for SciFact. This implies that the most important factor is not actually the size of the bin, but instead the size of the entry. Figure 10 (bottom left) shows that retrieval quality increases with the hash table size. For MS MARCO, MRR@10 and Recall@10 remain below 0.01 up to 100K and reach 0.042 and 0.054 at 1M, still increasing at the largest tested size. For SciFact, MRR@10 and Recall@10 increase up to 50K, reaching 0.45 and 0.53, and see no notable benefit.

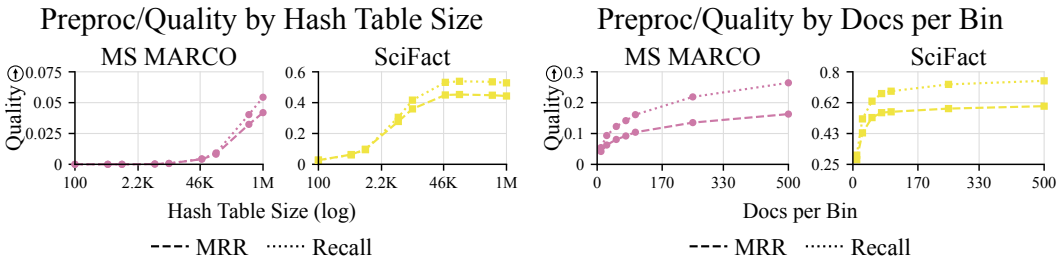


Figure 10: `PILLAR-Bin` PIR preprocessing time (top) and retrieval quality (bottom). Left: varying the hash table size. Right: varying the number of documents per bin.

1 or 2-round PIR `PILLAR-Bin` can store the document embeddings directly in the bins and return them with a single PIR query (Single DB), or store document IDs in the bins and retrieve the corresponding embeddings from a separate database with a second PIR query (Separate Embedding DB). Figure 11 shows that Single DB has lower or comparable WAN latency, LAN latency, compute and data sent at every tested number of documents per bin, and that the gap grows with the number of documents per bin. The difference is largest for SciFact: at 500 documents per bin, Single DB reduces WAN latency from 1.2s to 240ms, LAN latency from 235ms to 77ms, compute from 110ms to 7ms, and data sent from 19MB to 7.8MB. Single DB is usually cheaper because it removes the second round of PIR, the most expensive component of the protocol. Future protocols should always aim to minimize PIR rounds.

C Protocol Details

This section gives the detailed protocol figures for `PILLAR-Bin` and `PILLAR-Tree`, together with some implementation details that Section 4 suppresses for clarity. The first concerns what a RAG pipeline actually returns. A RAG pipeline consumes document *text*, whereas the dense stage of `PILLAR-Bin` outputs embeddings, and an embedding does identify the document it came from. PACMANN and `PILLAR-Tree` are unaffected, since both retrieve document indices in their traversals, but `PILLAR-Bin` does not. Each entry of the database held by `srv` in the `PILLAR-Bin` protocol stores an index alongside the embedding, using an extra 32 bits per row, and the protocol outputs indices that `clnt` uses in the final PIR round of protocol 15. Figure 12 gives the construction in full.

There is another concern regarding storage used by the client. As described in Section 4, the database in `PILLAR-Bin` doesn't store one document per row, like PACMANN or `PILLAR-Tree`, but instead t document embeddings per row. Under a PIR scheme with preprocessing, `clnt` retains a collection of entries in the database, that scale sub-linearly with the database size, so this falls on client storage rather than on the server alone (see Appendix B.1). We therefore provide a second variant that splits retrieval into two rounds, the first recovers identifiers from a database of 32-bit sized rows, and the second recovers the corresponding embeddings from a database holding each document embedding once. Figure 13 gives this variant, which trades one additional round for a client state independent of R and t.

Figure 14 presents the formal construction of `PILLAR-Tree` protocol. Finally, Figure 15 shows the whole PPRAG pipeline using the `PILLAR-Bin`/`PILLAR-Tree` constructions.

D Complexity Analysis

D.1 Notation

We denote the cost of a batch PIR call on ψ rows over a database of $\mathcal{R}$ rows, each row consist of C fields as $\texttt{PIRCost}(\psi, \mathcal{R}, C)$. For Batch PianoPIR implementation from PACMANN Zhou et al. (2024b), the cost is $\widetilde{O}(C\sqrt{\mathcal{R}\psi})$

Protocol 1.a: `PILLAR-Bin` (single-round variant)

Participants. Server `srv` and client `clnt`.

Public parameters. Corpus $\mathcal{C} = \{D_1, \ldots, D_N\}$; analyzer Analyze; embedding function Emb of dimension d; BM25 parameters (k_1, b); row count R; row depth t; hash function $H : \mathcal{V} \to \{0, \ldots, R-1\}$; batch bound β; output size k; reserved dummy index $\perp$.

Client input. Query Q with $\mathsf{Analyze}(Q) = (w_1, \ldots, w_{|Q|})$.

Preprocessing (server `srv`).

1. `srv` initialize empty database DB of R rows.
2. For each $w \in \mathcal{V}$, `srv` scores every $D \in \mathcal{C}$ containing w under BM25 with w as a single-term query, and sets $\mathcal{L}_w$ to the t highest-scoring documents in descending order, padded with $\langle \perp, \mathbf{0} \rangle$ if fewer than t documents contain w. Write $\mathcal{L}_w[j]$ for its jth document.
3. `srv` groups terms by row: w belongs to row $z = H(w)$. Write $W_z = \{w \in \mathcal{V} : H(w) = z\}$ and let $m_z = |W_z|$ be the number of terms that collide in row z, and fix an arbitrary order $w^{(1)}, \ldots, w^{(m_z)}$ on them, so that $\mathcal{C}_z = \{\mathcal{L}_{w^{(1)}}, \ldots, \mathcal{L}_{w^{(m_z)}}\}$ are the ordered lists competing for row z.
4. `srv` fills row z by interleaving the lists of $\mathcal{C}_z$ round-robin, one rank at a time: the jth document of the ith list goes to position $(j-1)m_z + i$: $\mathsf{DB}[z][(j-1)m_z + i] \leftarrow \mathcal{L}_{w^{(i)}}[j]$. The row therefore holds the best document of every colliding term, then the second-best of every term, and so on.
5. `srv` discards duplicates, which arise when a document appears in two colliding lists, and then truncates each row to its first t documents. A row with $m_z = 0$ is left empty.
6. `srv` then replaces all document text with embeddings and an index: For every row index $0 < z \leq R$ and offset $0 < x \leq t$, `srv` retrieves $D = \mathsf{DB}[z][x]$, computes $e_D = \mathsf{Emb}(D)$ and sets $\mathsf{DB}[z][x] = \langle \mathsf{id}(D), e_D \rangle$.
7. `srv` runs the PIR preprocessing phase over $\mathsf{DB}_{\mathsf{Emb}}$ with `clnt`, who stores the resulting private state.

Retrieval.

1. `clnt` computes $\mathsf{Analyze}(Q)$ and the row indices $r_i \leftarrow H(w_i)$ for $i \leq |Q|$, removing repeated indices.
2. `clnt` fixes the batch to exactly β indices and if $|Q| > \beta$ it retains the first β or if $|Q| < \beta$ they pad with random indices from $\{0, \ldots, R-1\}$.
3. `clnt` sends a single batched PIR query for these β indices and recovers the corresponding β rows of DB from `srv`'s response and its private state.
4. `clnt` locally discards entries with index $\perp$ and duplicate indices, forming a candidate pool $\mathcal{D}_{\text{cand}}$ of at most βt embeddings.
5. `clnt` computes $\mathbf{e}_Q = \mathsf{Emb}(Q)$ locally and ranks $\mathcal{D}_{\text{cand}}$ by $\cos(\mathbf{e}_Q, \mathbf{e}_D)$.
6. `clnt` outputs the k document indices of highest similarity to the query.

Figure 12: Formal specification of `PILLAR-Bin` (single-round variant). `srv` observes exactly β batched PIR queries, and holds a database DB of R rows. Each row is an ordered list of t entries and an entry is a pair $\langle \mathsf{id}(D), \mathsf{Emb}(D) \rangle$ of a 32-bit index and a d-dimensional vector.

D.2 PILLAR-BIN COMPLEXITY

Round complexity. There are two variants of `PILLAR-Bin`. For the first variant in Figure 13, the client runs two rounds of communication: The first round to retrieve $|Q|$ rows from the database, the second round to retrieve the document embedding corresponding to the document indices retrieved

Protocol 1.b: `PILLAR-Bin` (two-round variant)

Participants. Server `srv` and client `clnt`.

Public parameters. Identical to Protocol 1.a, with a second batch bound β'.

Client input. Query Q with $\mathsf{Analyze}(Q) = (w_1, \ldots, w_{|Q|})$.

Preprocessing (server `srv`).

1. `srv` builds DB exactly as in Protocol 1.a preprocessing, steps 1–4.
2. `srv` replaces all document text with a unique index only. For every row index $0 < z \leq R$ and offset $0 < x \leq t$, `srv` retrieves $D = \mathsf{DB}[z][x]$ and sets $\mathsf{DB}[z][x] = \mathsf{id}(D)$, the index alone.
3. For each $D_j \in \mathcal{C}$, `srv` initializes a database $\mathsf{DB}_{\mathsf{Emb}}$ by writing $e_D = \mathsf{Emb}(D_j)$ to row j of $\mathsf{DB}_{\mathsf{Emb}}$.
4. `srv` runs the PIR preprocessing phase over DB and over $\mathsf{DB}_{\mathsf{Emb}}$ with `clnt`, who stores the resulting private states.

Retrieval.

1. `clnt` computes retrieval steps 1–2 of Protocol 1.a exactly.
2. `clnt` sends one batched PIR query to DB, and recovers β rows of indices.
3. `clnt` locally discards $\perp$ and duplicate indices, obtaining a set I of, at most, $\beta \cdot t$ distinct indices.
4. `clnt` fixes I to exactly β' indices: if $|I| > \beta'$ it retains the first β' indices, if $|I| < \beta'$ it appends indices drawn randomly from $\{1, \ldots, R\}$.
5. `clnt` sends a single batched PIR query using these β' indices to $\mathsf{DB}_{\mathsf{Emb}}$ and recovers the corresponding embeddings.
6. `clnt` computes $\mathbf{e}_Q = \mathsf{Emb}(Q)$ locally and ranks the documents of I by $\cos(\mathbf{e}_Q, \mathbf{e}_D)$, ignoring the embeddings returned for padding indices.
7. `clnt` outputs the k document indices of highest similarity to the query.

Figure 13: Formal specification of `PILLAR-Bin` (two-round variant). The `srv` stores two databases, one database DB of R rows of t indices each, and a database $\mathsf{DB}_{\mathsf{Emb}}$ of N rows, row j holding the single embedding $\mathsf{Emb}(D_j)$. The primary difference is that the server only has to hold one embedding per entry, instead of t embeddings per entry.

in the first round. For the second variant in Figure 12, `clnt` run a single round and retrieve the embeddings directly from `srv` and rank those $|Q| \times t$ retrieved embeddings locally.

PIR cost. The number of rows being fetched are β for the first round over a total of R rows, where each row has an average of $O(|\mathcal{V}|t/R)$ document IDs, and $O(\beta' \times t)$ rows for the second round over a database of N rows, each with d fields. Thus, the total cost of the `PILLAR-Bin` (2-round) is: $\boxed{\texttt{PIRCost}(\beta, R, |\mathcal{V}|t/R) + \texttt{PIRCost}(t\beta', N, d)}$.

For the `PILLAR-Bin` (1-round), the complexity is simply β queries over a database of R rows, each with t document embeddings, resulting in total PIR cost of $\boxed{\texttt{PIRCost}(\beta, R, td)}$

Local computation. The client locally computes cosine distance between the query embedding and all retrieved document embeddings, incurring $O(|Q|td)$ computation complexity at client.

Protocol 1: BM25-Tree — Privacy-Preserving Lexical Pruning with Dense Ranking

Participants. Server `srv` and client `clnt`

Public Parameters. Corpus $\mathcal{C}$ of N documents:$\mathcal{C} = \{D_1, \ldots, D_N\}$; $\mathcal{V}$ be set of all unique terms within all the documents in $\mathcal{C}$; leaf block size B, sub-block size s, tree branching factor r, beam width W, $\mathcal{T}$ be set of all tree nodes, $\mathcal{T}_\ell$ be set of tree nodes on level ℓ, stage 2 number of retrieved sub-block k_s, maximum unique query terms $q^\star$, m public hash functions $h_1, \ldots h_m : \mathcal{V} \times \mathcal{T} \to \{0,1\}^\star$; Cuckoo Hash table load factor α ; target result size k;

Client Input. Query $Q = \{w_1, \ldots, w_{q^\star}\}$.

Server state (initialized).

1. Let $\mathcal{H} = 1 + \lceil \log_r(N/B) \rceil$ be height of r-ary tree. For each level $\ell \in [\mathcal{H}]$, `srv` computes N_ℓ the number of pairs $(w, \mathcal{B}) \in \mathcal{V} \times \mathcal{T}_\ell$ such that $\sigma_w(\mathcal{B}) \neq 0$. `srv` then initializes a database $\mathsf{DB}_{tree,\ell}$ of $\lceil N_\ell/\alpha \rceil$ rows for each $\ell \in [\mathcal{H}]$.
2. `srv` initialize an empty database DB_{vector} of $\lceil N/s \rceil$ rows.

Preprocessing (server `srv`)

1. For each node $\mathcal{B}_{i,\ell}$ in level ℓ for $1 \leq \ell \leq \mathcal{H} - 1$
   (a) `srv` computes $\sigma_w(\mathcal{B}_{i,\ell+1,j})$ for child node $\mathcal{B}_{i,\ell+1,j}$ of $\mathcal{B}_{i,\ell}$ ($j \in [r]$) and all words w that appears in documents in $\mathcal{B}_{i,\ell}$.
   (b) For each word $w \in \mathcal{B}_{i,\ell}$, the tuple $(w, \mathsf{id}(\mathcal{B}_{i,\ell}), \sigma_w(\mathcal{B}_{i,\ell+1,1}), \ldots, \sigma_w(\mathcal{B}_{i,\ell+1,r}))$ of $r+2$ fields are added to the database $\mathsf{DB}_{tree,\ell}$ using Cuckoo Hash rule on input $(w, \mathsf{id}(\mathcal{B}_{i,\mathcal{H}}))$ over m hash functions $h_1, \ldots, h_m$.
2. For each leaf block $\mathcal{B}$, `srv` computes the score of their sub-blocks $\mathcal{SB}_1, \ldots, \mathcal{SB}_{\lceil B/s \rceil}$ of size s with respect to each word w appearing in the block, store $(w, \mathsf{id}(\mathcal{B}), \sigma_w(\mathcal{SB}_1), \ldots, \sigma_w(\mathcal{SB}_{\lceil B/s \rceil}))$ to $\mathsf{DB}_{tree,\mathcal{H}}$ using Cuckoo Hash on input $(w, \mathsf{id}(\mathcal{B}))$ and m hash functions $h_1, \ldots, h_m$.
3. For each sub-block $\mathcal{SB}$ at position $p = \mathsf{id}(\mathcal{SB})$, store at row p of DB_{vector} the tuple $(e_{D_1}, \ldots, e_{D_s})$ the embedding vectors of s documents $D_1, \ldots, D_s$ within $\mathcal{SB}$.

Figure 14: The BM25-Tree protocol for privacy-preserving lexical pruning with dense re-ranking (Part I: setup and preprocessing).

D.3 PILLAR-TREE COMPLEXITY

Round Complexity. The client runs a total of $\mathcal{H} + 1$ rounds: First $\mathcal{H} - 1$ rounds to perform interactive tree traversal, 1 round for sub-block retrieval, and 1 round for embedding retrieval.

PIR cost. (1) For tree traversal, the number of rows being fetched on level ℓ is $mWq^\star$ over a database of $\lceil N_\ell/\alpha \rceil$ rows, each row consist of $r+2$ fields, resulting a cost of $\texttt{PIRCost}(mWq^\star, \lceil N_\ell/\alpha \rceil, r+2)$. (2) For sub-block retrieval, the client fetches $mWq^\star$ over a vector of $\lceil N_\mathcal{H}/\alpha \rceil$ rows each with $\lceil B/s \rceil + 2$ fields, resulting in a $\texttt{PIRCost}(mWq^\star, \lceil N_\mathcal{H}/\alpha \rceil, \lceil B/s \rceil + 2)$. (3) For embedding retrieval, the client fetches k_s rows over DB_{vector} of $\lceil N/s \rceil$ rows, each row has sd fields, resulting $\texttt{PIRCost}(k_s, \lceil N/s \rceil, sd)$ cost.

Hence, the total PIR cost of the `PILLAR-Tree` protocol is

$$\begin{aligned}\texttt{PIRCost}_{tree} = &\sum_{\ell=1}^{\mathcal{H}-1} \texttt{PIRCost}(mWq^\star, \lceil N_\ell/\alpha \rceil, r+2) \\ &+ \texttt{PIRCost}(mWq^\star, \lceil N_\mathcal{H}/\alpha \rceil, \lceil B/s \rceil + 2) \\ &+ \texttt{PIRCost}(k_s, \lceil N/s \rceil, sd)\end{aligned}$$

Local computation. The client local computation involves computing block score at each level between the input query and retrieved blocks' children. This costs $O(mWrq^\star)$ at each level from $\ell = 2, \ldots, \mathcal{H} - 1$ (the first level has only 1 node so it does not require any computation). At sub

Protocol 1: BM25-Tree (continued)

Stage 1: Tree Traversal

1. clnt initializes $F_0 = \{\mathcal{B}_{1,0}\}$, where $\mathcal{B}_{1,0}$ is the root node of the tree.
2. For each level $\ell = 0, \ldots, \mathcal{H} - 1$ do
   (a) Let $F_\ell = \{\mathcal{B}^\star_{1,\ell}, \ldots, \mathcal{B}^\star_{|F_\ell|,\ell}\}$ be the set of candidate nodes at level ℓ. For each $w \in Q$, clnt sends PIR to srv, retrieving rows $\big(h_a\big(w, \mathsf{id}(\mathcal{B}^\star_{i,\ell})\big)\big)_{\substack{a\in[m]\\ i\in[|F_\ell|]}}$ from $\mathsf{DB}_{tree,\ell}$, for a total of $m \times |F_\ell|$ rows.
   (b) For each $i \in [|F_\ell|]$, clnt checks whether any retrieved row has its first two fields equal to w and $\mathsf{id}(\mathcal{B}^\star_{i,\ell})$. If none does, clnt sets $\sigma_w(\mathcal{B}_{i,\ell+1,j}) = 0$, for all $j \in [r]$, where $\mathcal{B}_{i,\ell+1,j}$ is the j-th child of $\mathcal{B}^\star_{i,\ell}$. Otherwise, clnt sets $\sigma_w(\mathcal{B}_{i,\ell+1,j})$ to the $(j+2)$-th field of the matching row.
   (c) clnt computes $\sigma_Q(\mathcal{B}_{i,\ell+1,j}) = \sum_{w\in Q} \sigma_w(\mathcal{B}_{i,\ell+1,j})$ for every child of every node in F_ℓ.
   (d) clnt locally ranks these child nodes according to σ_Q and lets $F_{\ell+1}$ be the set of the top $\min\{W, |\mathcal{T}_{\ell+1}|\}$ nodes.
3. clnt outputs $F_\mathcal{H} = \{\mathcal{B}^\star_{1,\mathcal{H}}, \ldots, \mathcal{B}^\star_{W,\mathcal{H}}\}$ as top W leaf block.

Stage 2: Sub-Block Retrieval

1. For each $w \in Q$ do
   (a) clnt sends PIR to srv, retrieving rows $(h_{i\in[m]}(w, \mathsf{id}(\mathcal{B}^\star_{1,\mathcal{H}})), \ldots, h_{i\in[m]}(w, \mathsf{id}(\mathcal{B}^\star_{W,\mathcal{H}})))$ ($m \times W$ rows total) from $\mathsf{DB}_{tree,\mathcal{H}}$
   (b) For each $i \in [W]$, clnt checks if any of the retrieved data has the first two field equal to w and $\mathsf{id}(\mathcal{B}^\star_{i,\mathcal{H}})$. If there are non, clnt sets $\sigma_w(\mathcal{SB}_{i,\mathcal{H},j}) = 0$ for all sub-block $\mathcal{SB}_{i,\mathcal{H},j}$ of $\mathcal{B}_{i,\mathcal{H}}$, else clnt matches corresponding $\sigma_w(\mathcal{SB}_{i,\mathcal{H},j})$ with the $j+2$-th retrieved value of the corresponding row matches $(w, \mathcal{B}_{i,\mathcal{H}})$.
2. clnt computes $\sigma_Q(\mathcal{SB}_{i,\mathcal{H},j}) = \sum_{w\in Q} \sigma_w(\mathcal{SB}_{i,\mathcal{H},j})$ for all sub-blocks of top-W leaf block.
3. clnt locally rank and get set of top k_s sub-blocks $\mathcal{SB}^\star_1, \ldots, \mathcal{SB}^\star_{k_s}$

Stage 3: Dense Retrieval and Ranking

1. clnt sends PIR query to srv, retrieving rows $(\mathsf{id}(\mathcal{SB}^\star_1), \ldots, \mathsf{id}(\mathcal{SB}^\star_{k_s}))$ from DB_{vector}
2. Let $\mathcal{D}_{cand} = \{D^\star_1, \ldots, D^\star_{k_s\times s}\}$ be set of $k_s \times s$ documents whose embedding vectors are retrieved. clnt locally computes $\cos(e_Q, e_{D^\star_j})$ for all $j \in [k_s \times s]$ and ranks the score
3. Client output top k documents in $\mathcal{D}_{cand}$ with highest similarity.

Figure 14: The BM25-Tree protocol for privacy-preserving lexical pruning with dense re-ranking.

block level, client performs the subblock score computation, costing $O(mW\lceil B/s\rceil q^\star)$. Finally, the client computes the cosine similarity between his query and $k_s \times s$ retrieved document embeddings, resulting in $O(k_s s d)$ computation.

Thus, the client local computation costs $\boxed{O(mWr\mathcal{H}q^\star + mW\lceil B/s\rceil q^\star + k_s s d)}$

E Detailed Security Analysis

We prove that both protocols provide security against a semi-honest server. Security is formally defined via the standard simulation paradigm.

We first define computationally private PIR ideal functionality as:

Definition 1 (Computationally Private PIR (Kushilevitz and Ostrovsky, 1997; Chor and Gilboa, 1997)). *A single-server PIR scheme* $\Pi_{PIR} = (\mathsf{Query}, \mathsf{Answer}, \mathsf{Decode})$ *over a database* DB *of* N *records satisfies:*

Protocol 3: Full PPRAG based on `PILLAR`

Participants. Server `srv` and client `clnt`.

Public parameters. Corpus $\mathcal{C}$ of N documents:$\mathcal{C} = \{D_1, \ldots, D_N\}$, Set params of parameters for `PILLAR` (`PILLAR-Bin`/`PILLAR-Tree`) retrieval algorithm.

Client input. Keyword query $Q = \{w_1, \ldots, w_{|Q|}\}$.

Server state (initialized / preprocessing).

1. `srv` runs `PILLAR` initialization on params
2. `srv` runs `PILLAR` preprocessing on $\mathcal{C}$, params
3. `srv` and `clnt` preprocess $\mathcal{C}$ for PIR, each document $D_i \in \mathcal{C}$ associate with an identifier $i \in [N]$.

Retrieval

1. `clnt` and `srv` participate in private retrieval `PILLAR` protocol on Q and $\mathcal{C}$, `clnt` receive indices top-k documents in $\mathcal{C}$ with respect to Q. Call the set $\mathsf{ID}_{top-k} = \{D_1^\star, \ldots, D_k^\star\}$
2. `clnt` runs PIR to retrieve content of k documents $D_1^\star, \ldots, D_k^\star$.

Generation

`clnt` runs local LLM to generate answer with input query Q and k documents $D_1^\star, \ldots, D_k^\star$, output answer $\mathcal{A}$

Figure 15: `PILLAR`-based PPRAG protocol. `clnt` and `srv` participates in online secure retrieval before `clnt` runs inference on its local LLM.

1. ***Correctness:*** *For any index $i \in [N]$:*

$$\Pr[\mathsf{Decode}(\mathsf{Answer}(DB, \mathsf{Query}(i))) = DB[i]] \geq 1 - \mathsf{negl}(\lambda).$$

2. ***Computationally Private:*** *for all indices $i, j \in [N]$ and all PPT distinguishers $\mathcal{D}$:*

$$\left|\Pr\left[\mathcal{D}(\mathsf{Query}_\Pi(i)) = 1\right] - \Pr\left[\mathcal{D}(\mathsf{Query}_\Pi(j)) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Before stating the main theorems, we establish a composition lemma that underpins the security of `PILLAR-Tree`'s multi-round interaction.

Lemma 2 (Sequential PIR Composition)**.** *Let Π_{PIR} be a computationally private PIR scheme. Let $T = T(\lambda)$ be a polynomial. Any protocol that issues a sequence of T independent PIR queries, where T is determined solely by public parameters, remains computationally private against a semi-honest server.*

Proof. Define a sequence of hybrid experiments $\mathsf{H}_0, \mathsf{H}_1, \ldots, \mathsf{H}_T$ where H_ℓ denotes the distribution in which the first ℓ PIR queries are replaced by simulated queries $\mathsf{Query}_\Pi(u_i)$ for independently uniform $u_i \xleftarrow{\$} [N]$, while the remaining $T - \ell$ queries are real. H_0 is the real server view; H_T is the fully simulated view.

By Definition 1, for each $\ell \in [T]$ and every PPT distinguisher $\mathcal{D}$:

$$\left|\Pr[\mathcal{D}(\mathsf{H}_{\ell-1}) = 1] - \Pr[\mathcal{D}(\mathsf{H}_\ell) = 1]\right| \leq \mathsf{negl}(\lambda).$$

Summing over $T = \mathrm{poly}(\lambda)$ hybrids, the total distinguishing advantage is at most $T \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$, completing the proof. □

Theorem 1 (Security of `PILLAR-Bin` (1-round))**.** *Suppose Π_{PIR} is a computationally private PIR scheme (Definition 1). Then Protocol 1 (`PILLAR-Bin`, Figure 12) is secure against a semi-honest server.*

Proof. **Server's real view.** The server `srv` performs all preprocessing locally and therefore already knows the full hash table. During retrieval, `srv`'s view consists solely of the β PIR queries $\{\mathsf{Query}_\Pi(r_i)\}_{i=1}^{\beta}$ issued by the client, where $r_i = H(w_i) \bmod R$ for keyword $w_i \in \mathcal{V} \cup \{\perp\}$ in the padded/truncated Q. No other message passes from `clnt` to `srv`.

Simulation. Construct Sim as follows: given only the public parameters (N, R, k, λ) and the fixed query length β, output $\{\mathsf{Query}_\Pi(u_i)\}_{i=1}^{\beta}$ where each $u_i \xleftarrow{\$} [R]$ is independently uniform.

Indistinguishability. The simulated view consists of β uniformly random PIR queries; the real view consists of β PIR queries for indices $r_1, \ldots, r_\beta \in [R]$ determined by the query keywords. By Lemma 2 (applied with database size R and $T = \beta$ queries), the real and simulated views are computationally indistinguishable, completing the proof. □

Theorem 2 (Security of `PILLAR-Bin` (2-round))**.** *Suppose $\Pi_{PIR}^{(1)}, \Pi_{PIR}^{(2)}$ is a computationally private PIR scheme (Definition 1) on the index database DB and the vector database $\mathsf{DB}_{\mathsf{Emb}}$ respectively. Then Protocol 1 (`PILLAR-Bin`, Figure 13) is secure against a semi-honest server.*

Proof. **Server's real view.** The server `srv` performs all preprocessing locally and therefore already knows the full hash table. During retrieval,

- **First round:** `srv`'s view consists solely of the β PIR queries $\{\mathsf{Query}_\Pi^{(1)}(r_i)\}_{i=1}^{\beta}$ issued by the client, where $r_i = H(w_i) \bmod R$ for keyword $w_i \in \mathcal{V} \cup \{\perp\}$ in the padded/truncated Q.
- **Second round:** `srv`'s view consists of β' queries $\{\mathsf{Query}_\Pi^{(2)}(\zeta_i)\}_{i=1}^{\beta'}$ where $\zeta_i \in I^\star$ [6] to the embedding database $\mathsf{DB}_{\mathsf{Emb}}$ issued by the client, where β' is a known public parameter.

No other message passes from `clnt` to `srv`. We note that de-duplication and removal of $\perp$ on retrieval step 3 of Figure 13 are happening locally at client for internal score ranking, and `clnt` still sends a constant payload of β' queries in step 4-5 regardless of deduplication/removal results of step 3.

Simulation. Construct Sim as follows: given only the public parameters (N, R, k, λ) and the fixed query lengths β, β', output $\{\mathsf{Query}_\Pi^{(1)}(u_i)\}_{i=1}^{\beta}, \{\mathsf{Query}_\Pi^{(2)}(z_i)\}_{i=1}^{\beta'}$ where each $u_i \xleftarrow{\$} [R], z_i \xleftarrow{\$} [N]$ are independently uniform.

Indistinguishability. The simulated view consists of $\beta + \beta'$ uniformly random PIR queries; the real view consists of β PIR queries for indices $r_1, \ldots, r_\beta \in [R], \zeta_1, \ldots, \zeta_{\beta'} \in [N]$, determined by the query keywords. By Lemma 2 (applied with database size R and $T = \beta$ queries; database size N and $T = \beta'$ queries), the real and simulated views are computationally indistinguishable, completing the proof. □

Theorem 3 (Security of `PILLAR-Tree`)**.** *Suppose Π_{PIR} is a computationally private PIR scheme (Definition 1). Then Protocol 2 (`PILLAR-Tree`, Figure 14) is secure against a semi-honest server*

Proof. `PILLAR-Tree` proceeds in three sequential rounds; we analyze each in turn and then apply Lemma 2.

Round 1 (Block-Max Tree Traversal). At level $\ell \in \{1, \ldots, \mathcal{H}\}$ of the r-ary tree with height $\mathcal{H} = 1 + \lceil \log_r(N/B) \rceil$, the client expands the beam frontier $F_{\ell-1}$ of at most W nodes by fetching the block-max scores of all $|F_{\ell-1}| \cdot r \le W \cdot r$ children via PIR. Because W and r are public parameters, the number of PIR queries at each level is at most $W \cdot r$ regardless of the query content Q. The server therefore observes at most $\mathcal{H} \cdot W \cdot r$ PIR queries over a database of N_{nodes} tree nodes, where N_{nodes} and $\mathcal{H} \cdot L \cdot r$ are both fixed public quantities.

Round 2 (Sub-Block Retrieval). The client fetches block-max scores for exactly $Z = W \cdot (B/s)$ sub-blocks, a count determined entirely by public parameters (W, B, s). The server observes exactly Z PIR queries over the sub-block database.

[6] $I^\star$ denotes the padded/truncated version of I with β' items, as described in Retrieval Step 4, Figure 13

Round 3 (Dense Embedding Fetch). The client fetches exactly $k_s \cdot s$ document embeddings via PIR over the database of N documents. The count $k_s \cdot s$ is a fixed public parameter.

Sequential Composition. The total number of PIR queries across all three rounds is $T = \mathcal{H} \cdot W \cdot r + Z + k_s \cdot s$, which is a deterministic polynomial function of the public parameters. By Lemma 2 applied to this T-query sequence, the server's view across all rounds is computationally indistinguishable from a simulation issuing T uniformly random PIR queries, completing the proof. □

Remark 3 (Access-Pattern Privacy)**.** *A critical property enabling Theorems 1, 2, and 3 is that the* number *of PIR queries per round is a deterministic function of public parameters $(N, B, s, r, W, k_s, q^\star, \beta, \beta')$, independent of the query content Q. This eliminates access-pattern leakage: the server cannot distinguish any two queries of equal length $|Q| = q^\star$. By contrast, adaptive protocols such as PACMANN must pad short-circuit executions with dummy queries to hide early convergence, introducing both efficiency overhead and a potential timing side-channel if padding is imperfect.*

Remark 4 (Correctness of Algorithms)**.** *We follow the common practice of finding approximate nearest neighbor of modern framework like HNSW (Malkov and Yashunin, 2018) or FAISS (Johnson et al., 2019): The retrieval correctness are not formally analyzed but rather showed by empirically evaluate it using benchmark dataset (Aumüller et al., 2018). Thus, we show the correctness experimentally in Section 5 and omit the proof on theoretical correctness gap between* `PILLAR-Bin`*,* `PILLAR-Tree` *and the ideal top-k nearest neighbors search.*

F PIR PILLAR-TREE DATABASE VIA CUCKOO HASHING

Notation. With the term $\mathsf{Tree}(\mathcal{C})$ we denote the tree build upon corpus $\mathcal{C}$ while with the term $\mathcal{L}_\ell$ we denote the collection of superblocks of layer ℓ of the tree. We use $H = O(\log_r |\mathcal{C}|/|\mathcal{B}|)$ to denote the height of the tree. For ease of exposition, we may abuse the notation $\mathcal{B}$ for both the block and the super block in the tree. Let $\mathsf{id}(\mathcal{B}_\ell)$ denote the index of block $\mathcal{B}_\ell$ within level ℓ. The client can compute this value locally, i.e., to fetch, say, the 4th node of level ℓ, it simply sets $\mathsf{id}(\mathcal{B}_\ell) = 4$.

Storage Blowup. Recall the initial proposal from the main paper in which at each level ℓ, every super block $\mathcal{B} \in \mathcal{L}_\ell$ is associated with an array of $|\mathcal{V}|$ integers storing the upper bounds. Such an array is extremely sparse (most of its entries are 0) so this representation wastes a substantial amount of storage. Crucially, the cost is not confined to space, i.e., PIR performance scales with the size of the PIR database, which here is $O(|\mathcal{L}_\ell| \cdot |\mathcal{V}|)$ for level ℓ, so the same redundancy inflates query time as well. Summing over all levels, representing each level as a concatenation of $|\mathcal{L}_\ell|$ sparse arrays forces the server to store $O(\sum_\ell |\mathcal{L}_\ell| \cdot |\mathcal{V}|)$ integers, the vast majority of which are 0.

Proposed Optimization. In order to reduce the PIR database storage, for each level ℓ on the tree, we present its corresponding PIR database as a *dictionary* (specifically, a hash map) of key-value pairs where the key is the string $\mathsf{key} = w||\mathsf{id}(\mathcal{B}_\ell)$ and the value is $\mathsf{value} = \sigma_w(\mathcal{B}_\ell)||w||\mathsf{id}(\mathcal{B}_\ell)$. The dictionary stores an entry only for those blocks whose score is non-zero, i.e., $\sigma_w(\mathcal{B}_\ell) > 0$, which is precisely what eliminates the sparsity of the previous representation. This, however, means the client may issue a query on a combination of w and $\mathsf{id}(\mathcal{B}_\ell))$ that has no entry in the dictionary. Because our dictionary always returns *some* entry (it cannot signal that a key is absent) the client needs a way to tell a genuine answer from an unrelated entry it happened to collide with. Repeating $w \,\|\, \mathsf{id}(\mathcal{B}_\ell)$ inside value supplies exactly this check; the client compares the returned suffix against the key it queried, and interprets a mismatch as $\sigma_w(\mathcal{B}_\ell) = 0$. Concretely, on level ℓ of $\mathsf{Tree}(\mathcal{C})$, let N_ℓ be the number of dictionary entries such that $w \in \mathcal{V}, \mathcal{B}_\ell$ on level ℓ of the tree, and $\sigma_w(B_\ell) > 0$. Thus, we have $N_\ell \ll |\mathcal{L}_\ell| \cdot |\mathcal{V}|$. The hash map is implemented via a *Cuckoo Hash Table*, with load factor α, which has $O(1)$ search time. We use m hash functions $h_1, \ldots, h_m : \mathcal{V} \times \mathcal{L}_\ell \to [N_\ell/\alpha]$ to map dictionary entries to a table of N_ℓ/α slots such that each slot contains at most one key-value pair, and the entry $(w||\mathsf{id}, \sigma_w(\mathcal{B}_\ell)||w||\mathsf{id}(\mathcal{B}_\ell))$ is mapped to slot $h_i(w||\mathsf{id})$ for one $i \in \{1, \ldots, m\}$. For an empty row in Cuckoo table, the data is $(\perp, 0)$. At query time, suppose the client wishes to learn the score of a block $\mathcal{B}_c \in \mathcal{L}_\ell$ with respect to one of its own keywords w_c. It forms the key $\mathsf{key}_c = w_c \,\|\, \mathsf{id}(\mathcal{B}_c)$, issues m PIR queries for the candidate slots $h_1(\mathsf{key}_c), \ldots, h_m(\mathsf{key}_c)$, and receives m values. Applying the check described above, the client compares each returned suffix against key_c to verify if this word appears in the corresponding superblock.

Complexity. On layer ℓ of the tree, the new representation incurs $m \cdot w \cdot r \cdot |Q|$ queries on a database of size $N_\ell/\alpha \cdot 3$ (N_ℓ/α rows, three fields per row). On comparison, the old, unoptimized version has $w \cdot r \cdot |Q|$ queries on a database of size $|\mathcal{V}| \cdot |\mathcal{L}_\ell|$ (i.e., $|\mathcal{V}| \cdot |\mathcal{L}_\ell|$ rows with one field per row). In total over H levels, the storage size of the new representation is $\sum_{\ell=1}^{H} N_\ell/\alpha$ comparing to the old cost of $O(|\mathcal{V}| \cdot \frac{|\mathcal{C}|}{|\mathcal{B}|})$, where $\sum_{\ell=1}^{H} N_\ell \ll \sum_{\ell=1}^{H} |\mathcal{L}_\ell| \cdot |\mathcal{V}| = O(|\mathcal{V}| \cdot \frac{|\mathcal{C}|}{|\mathcal{B}|})$.

G Supplementary details

G.1 Full Testbed.

We detail the extra details regarding our testbed that would be too verbose for the main body: We also test LAN latency in Appendix B, which has 1000 megabits per second bandwidth and 5 millisecond round-trip-time. Every configuration was given 120 GB of RAM, with the exception of `PILLAR-Bin` configurations that: hold the embedding database in memory and have more than 1.5K documents per row, those were run on a machine with 1 TB of RAM. For SciFact's document/query embeddings we used 192-dimension embeddings created from the `e5-base-v2` (Wang et al., 2024) embedding function. For MS MARCO's document/query embeddings we used 192-dimension embeddings created from the `msmarco-MiniLM-L-6-v3` (The same as PACMANN did, to keep the comparison fair), Sentence-BERT bi-encoder (Reimers and Gurevych, 2019), built on MiniLM (Wang et al., 2020) and fine-tuned on MS MARCO. We ran our LLM/AI experiments on a single NVIDIA A100 (80GB VRAM) and fixed the RAGAS evaluation stage to the same 100 queries per dataset.

G.2 Hybrid Retrieval in Practice.

Combining lexical and dense retrieval is a standard, first-class feature of the search engines on which RAG pipelines are built. Elasticsearch exposes a reciprocal rank fusion (RRF) retriever (Cormack et al., 2009) that merges the rankings of a BM25 query and a k-NN query into a single result list (Elastic). OpenSearch introduced a dedicated hybrid query in version 2.11 that normalizes and combines keyword and neural scores inside the engine (OpenSearch Project). Weaviate runs vector and BM25 search in parallel and fuses the two result sets with a configurable weighting (Weaviate). Azure AI Search likewise executes keyword and vector queries in parallel and merges them with RRF, and its documentation reports that, in most benchmark tests, hybrid queries with semantic reranking return the most relevant results (Microsoft). A Microsoft whitepaper, drawing on production RAG applications serving billions of queries per day, describes hybrid search with reranking as "table stakes" for RAG (Berntson et al.). Model providers make the same recommendation: Anthropic's Contextual Retrieval pipeline pairs embeddings with BM25, and adding the BM25 component reduced the top-20 retrieval failure rate from 3.7% to 2.9% (Anthropic, 2024). These sources establish that hybrid retrieval is broadly supported and recommended in practice; they do not constitute a census of deployed systems, and many simple RAG pipelines remain dense-only.