Skyline Retrieval meets Set-Cover Chunk Merging: A Cost-Effective RAG-Sketch for Long-Context LLM QA

XINYI ZHU, The Hong Kong University of Science and Technology (Guangzhou), China
HAOYANG LI*, The Hong Kong Polytechnic University, China
YONGQI ZHANG, The Hong Kong University of Science and Technology (Guangzhou), China
LEI CHEN, The Hong Kong University of Science and Technology (Guangzhou) and The Hong Kong University of Science and Technology, China

Large Language Model (LLM) question answering (QA) is essential for various real-world applications. As the demand for LLMs in domain-specific, up-to-date, and private scenarios increases, equipping them with large external corpus for QA, referred to long-context LLM QA, becomes critical. However, existing methods struggle to balance the comprehensive retrieval of scattered information required for accurate QA with the high token costs of long-context processing. We identify two major causes of this bottleneck: (1) **limited retrieval coverage** under complex queries and large input scales, and (2) **repeated document access** enforced by rigid query-by-query processing. In this paper, we introduce a cost-effective Retrieval-Augmented Generation Sketch (RAG-Sketch) for long-context LLM QA. To enable more thorough information retrieval without exhaustive context traversal, we propose a new skyline retrieval module incorporating query decomposition into a pre-defined schema. This approach retrieves Pareto optimal chunks by jointly considering semantic similarity and the presence of key information within the decomposed queries. Based on retrieved chunks, we formally define a set cover chunk merging optimization problem and proposes a greedy algorithm with theoretical guarantees that effectively reduces redundant access. Our experiments on widely used long-context QA benchmarks demonstrate that RAG-Sketch improves prediction accuracy by 24.5% while reducing token costs by 58.5% compared with the state of the art.

CCS Concepts: • **Information systems → Data management systems**.

Additional Key Words and Phrases: Long-context Question Answering, Retrieval Augmented Generation

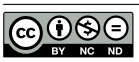
ACM Reference Format:
Xinyi Zhu, Haoyang Li, Yongqi Zhang, and Lei Chen. 2026. Skyline Retrieval meets Set-Cover Chunk Merging: A Cost-Effective RAG-Sketch for Long-Context LLM QA. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 234 (June 2026), 27 pages. https://doi.org/10.1145/3802111

1 Introduction

Large Language Models (LLMs) are deep learning models trained on massive amounts of data with billions of parameters [4, 31]. They take textual input (a question), referred to as a prompt or context, and generate textual output (an answer). Although LLMs have demonstrated remarkable capabilities across various natural language processing tasks [41, 44, 70, 71], they often lack specific knowledge

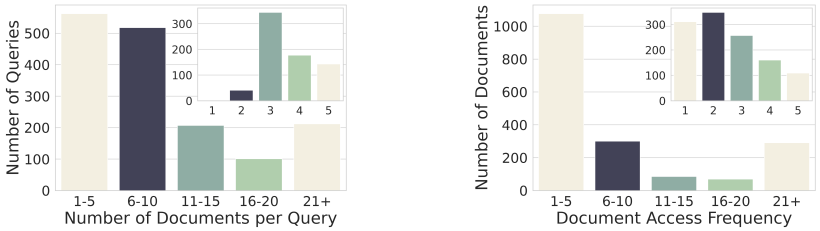
*Corresponding Author

Authors' Contact Information: Xinyi Zhu, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China, xzhu683@connect.hkust-gz.edu.cn; Haoyang Li, haoyang-comp.li@polyu.edu.hk, The Hong Kong Polytechnic University, Hong Kong SAR, China; Yongqi Zhang, yongqizhang@hkust-gz.edu.cn, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China; Lei Chen, leichen@cse.ust.hk, The Hong Kong University of Science and Technology (Guangzhou) and The Hong Kong University of Science and Technology, Guangzhou, China.



ACM 2836-6573/2026/6-ART234
https://doi.org/10.1145/3802111

in domain-specific, up-to-date, or private Question Answering (QA) scenarios [3, 14, 22, 28, 60, 66], as they are restricted to their pretrained data. Consequently, LLMs often require support from a local corpus [19, 32, 33, 59, 63]. In many real-world applications such as financial decision-making [3, 8, 23], legal analysis [58, 67], and research summarization [40], the corpus is large and multi-document (e.g., annual reports, contracts, research papers). QA tasks in these applications often require reasoning over dispersed evidence, such as year-over-year financial comparisons, cross-clause compliance checks, or multi-paper synthesis, which necessitate that existing QA systems support long-context input processing [32, 33, 59].



(a) # Documents Per Query (b) # Access Per Document

Fig. 1. Samples from Loong [59] Benchmark Dataset.

Equipping LLMs with such a large corpus for QA often termed *long-context LLM QA* [19, 32, 33, 59, 63] is challenging because the relevant evidence is sparse and distributed across very long inputs (tens or even hundreds of thousands of tokens) [7]. In the popular long-context QA Loong benchmark [59], over 31% queries reference more than 10 documents (Figure 1(a)). Although some modern LLMs can offer extremely large context windows (up to millions of tokens as input) [4, 56], feeding such long inputs naively introduces high computational overhead, API cost, and latency, and can dilute the model's attention over irrelevant content [6, 12, 14, 59].

As summarized in Figure 2, existing works follow three main paths to address the task: **Standard RAG**, which retrieves Top-K semantic similar chunks for generation [16, 28], **Long-context LLMs**, which enlarge the context window to include longer raw inputs [44, 59, 64], and **RAG with structured data**, which iterates over the corpus to extract structured representations like Knowledge Graphs (KGs) for subsequent retrieval and generation [14]. All three methods face a **cost–coverage wall** where fully covering the evidence needed to answer a query becomes prohibitively expensive in terms of input length to the model. Standard RAG reduces computational costs by retrieving only Top-K most relevant chunks for generation. However, a fixed K setting fails to capture all essential evidence across different tasks. On the Loong benchmark, RAG with various K settings exhibits a 25.4% performance drop compared to using all available information across four tasks [59]. In contrast, Long-context LLMs and RAG with structured data support higher information coverage by either extending the model context window or constructing structured representations over the corpus, yet this substantially increases the input costs [27, 59]. Specifically, building a KG for a 42,717-word document costs $7.83 with GPT-4o [14, 44]. We therefore ask whether this wall can be broken to deliver a cost-effective long-context LLM QA service.

To answer it, we identify two key factors that limit the cost-effectiveness. The first is the **limited retrieval coverage** under complex queries and large input scales. Queries in long-context QA are complex, invoking both information retrieval and subsequent reasoning [32, 33, 59]. For example, in financial analysis, a query "evaluate the company's recent health in noncurrent assets" [59] implicitly involves two steps: extracting all noncurrent asset values from previous financial reports, and then performing an analysis based on them. In addition, as shown in Figure 1(a), evidence for a

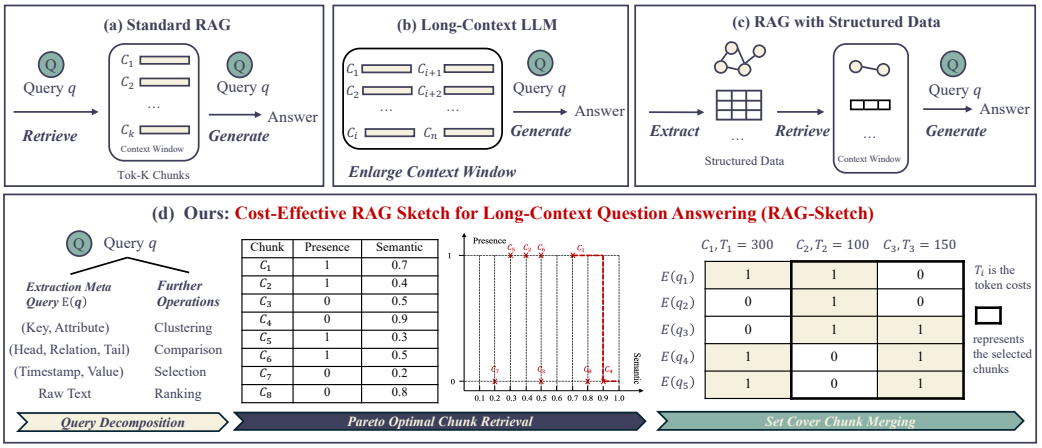


Fig. 2. An overview of existing approaches for the long-context QA task. Specifically, RAG-Sketch first decomposes each query for explicit processing steps. It then adaptively ensures complete retrieval of scattered information for more accurate QA without relying on a fixed top-K strategy as in standard RAG, or exhaustive document traversal as in long-context LLMs and structured RAG. Finally, RAG-Sketch merges meta-queries to minimize redundant access, thereby further reducing token costs.

single query is often scattered across multiple documents, with some pieces buried in low-salience locations (e.g., table cells or footnotes). Rather than relying on naive context traversal [14, 56], which incurs high computational costs to locate such evidence, conventional dense or sparse retrieval mechanisms still fail to ensure that all required information is covered within the Top-K results, causing long-context QA accuracy to plateau at approximately 41.6% [59]. The second is the **redundant document access** enforced by rigid query-by-query processing scheme in default LLM QA systems [14, 44, 59, 64, 73]. As shown in Figure 1(b), many documents are shared across multiple queries, with some accessed more than 20 times. Therefore, reusing shared chunks across queries can reduce repeated computation and improve overall system throughput. For instance, if both queries require information from the same chunk, it should ideally be accessed only once.

In this paper, we introduce RAG-Sketch, a novel RAG sketch designed to enhance cost-effectiveness of existing long-context LLM QA. Building on the two previously discussed observations, RAG-Sketch incorporates two innovative designs to tackle bottlenecks in existing long-context LLM QA methods. **First**, to enhance relevant information coverage despite large input scale and query complexity, RAG-Sketch introduces a skyline retrieval mechanism incorporating a reliable query decomposition into the pre-defined schema. As shown in Figure 2(d), RAG-Sketch firstly decomposes queries specifying extraction requirements, target data type, and further operations. The skyline retrieval mechanism then adaptively retrieves Pareto optimal chunks through a dual-dimensional partial order without relying on exhaustive context traversal or a fixed Top-K strategy. **Second**, to minimize redundant access costs while ensuring query coverage, RAG-Sketch proposes a set-cover chunk merging algorithm with bounds. In summary, this paper makes the following major contributions:

- We design a RAG-Sketch framework that enhances the cost-effectiveness of existing long-context LLM QA systems with a new adaptive skyline retrieval scheme and a set cover chunk merging mechanism.
- We propose an adaptive skyline retrieval algorithm for scattered information coverage that identifies Pareto optimal chunks, which are not dominated by any others, under a dual-dimensional

partial order that accounts for both semantic similarity and the presence of key information, thereby enabling comprehensive retrieval at reduced computational cost.
- We formally analyze and define the token costs minimization for retrieved chunks as an optimization problem, minimizing the access frequency while guaranteeing query coverage. We prove the problem is NP-hard, and propose a set cover chunk merging greedy algorithm with a theoretical guarantee.
- Our extensive experiments demonstrate that the proposed RAG-Sketch framework improves prediction accuracy by 24.5% while reducing costs by 58.5% compared with the state of the art. Furthermore, the cost-effectiveness evaluation, ablation studies, analyses with various backbone models, and generalization ability evaluation further validate RAG-Sketch's robustness.

2 Preliminary and Definition

This section formalizes the core concepts of LLM-based QA, with a particular focus on long-context scenarios. Important notations of this paper are summarized in Table 1.

We consider a LLM with the default context window length of W tokens. Let $\tau(\cdot)$ denote a tokenizer and $\ell(\cdot)$ the token-length function induced by τ. We reserve $h \geq 0$ tokens for system instructions and formatting. Then, the effective maximum per-query input length budget is $L_{\max} = W - h$.

Let $Q = \{q_j\}_{j=1}^{n}$ be a set of user queries. Let the external corpus be $\mathcal{D} = \{d_t\}_{t=1}^{m}$, where each document d_t is a token sequence with length $\ell(d_t)$. We write the total corpus length as $\ell(\mathcal{D})_{\text{total}} = \sum_{t=1}^{m} \ell(d_t)$.

Definition 2.1 (Closed-book LLM QA). Given $Q = \{q_j\}_{j=1}^{n}$, closed-book QA answers each query using only the model's parametric knowledge without external documents as $A_j = \text{LLM}(q_j)$.

In domain-specific, up-to-date, and private scenarios, closed-book LLM QA is insufficient. To address this limitation, existing works supplement LLMs with external documents. However, due to constraints in the input context window, it is necessary to first define the concept of a chunk and its associated token costs.

Definition 2.2 (Token length). For any text sequence X, its token length is $T(X) = \ell(X)$ produced by the tokenizer.

Definition 2.3 (Chunk). Given a document d_t, $\tau(d_t) = (w_1, \ldots, w_{L_t})$ is its token sequence. A *chunk* is any *contiguous* token subsequence

$$C = (w_a, \ldots, w_b) \quad \text{with } 1 \leq a \leq b \leq \ell(d_t),$$

returned by a (user-specified) chunking function $\sigma(\cdot)$ (e.g., fixed-size, sliding-window with stride, semantic segmentation, or document-aware rules). We denote the multiset of chunks for d_t by $\mathcal{C}(d_t) := \sigma(d_t)$ and the corpus-level chunk multiset by $\mathcal{C} = \biguplus_{t=1}^{m} \mathcal{C}(d_t)$.

Definition 2.4 (Chunk cost). The cost (length) of a chunk C is its token length $T(C)$. Since C is already in token space, $T(C)$ equals the number of tokens in C.

Definition 2.5 (Selected Chunk Set). For a query q, a budget-feasible context is any finite ordered multiset of chunks $S \subseteq \mathcal{C}$ such that $\ell(S) \leq L_{\max}$.

Specifically, the selected chunk set S described in Definition 2.5 is typically indexed and retrieved from chunks via semantic similarity in RAG framework.

Problem 1 (Long-Context LLM QA). *Given the document list $\mathcal{D}$, query batch Q and the LLM effective maximum context length L_{max}, the task answers each $q_j \in Q$ by first retrieving the chunk set*

Table 1. Important Notations

Notation	Definition
Q	Input Query Set
q_j	The j-th query in the query set $q_j \in Q$
A_j	The generated answer for q_j
$\mathcal{D}$	Documents in the local corpus
$L_{\max}$	The effective input context length of LLMs
C_i	The i-th divided chunk
$\mathcal{M}(q)$	The decomposed query tuple
$\mathrm{sim}(\cdot)$	The cosine similarity between vectors
$\phi(C_i)$	The value of C_i's presence dimension
$\omega(C_i)$	The value of C_i's semantic dimension
$T(C_i)$	The token cost of C_i
$E(q)$	The decomposed extraction meta-query
$\mathcal{A}_i$	The abstract of chunk C_i
$S(q)$	The selected Pareto optimal chunk set for q
F	The total costs of all selected chunks
$\boldsymbol{Q}(\cdot)$	The mapping function
M	The covered queries in each iteration
$G(\cdot)$	The objective function for chunk merge
$\mathbb{M}$	The final selected chunk list

$S(q_j)$ for the query q_j from $\mathcal{D}$ (Definition 2.5), and then calling the LLM:

$$A_j \;=\; LLM\left(q_j, S(q_j)\right).$$

An instance is called long-context *if the following holds: the minimal sufficient evidence for q_j spans multiple documents which is larger than L_{max} without selection or compression.*

In this paper, we aim to address the long-context LLM QA defined in Problem 1, subject to the following limitations as discussed in Section 1: (1) limited retrieval coverage, and (2) repeated document access. We aim to address these challenges by proposing a cost-effective RAG-Sketch, discussed in detail as follows.

3 Methodology

In this section, we present an overview of RAG-Sketch framework. We then elaborate on details of each module within RAG-Sketch.

3.1 Framework Overview

Figure 2(d) presents an overview of RAG-Sketch. The inputs are an external document list $\mathcal{D}$ and a query batch Q. The objective is to address Problem 1, which is to derive an answer A_j for each query $q_j \in Q$ according to the external documents. We summarize RAG-Sketch pipeline in Algorithm 1.
Query Decomposition. As shown in Algorithm 1 lines 1-2, given the query batch Q, RAG-Sketch instructs the LLM through a schema-constrained prediction to decompose each query $q \in Q$ into pre-defined schemas with high reliability as $\mathcal{M}(q) = (E(q), d, o)$: (1) the extraction meta-query $E(q)$; (2) the intended data type d; (3) the subsequent query operation o.
Adaptive Skyline Retrieval. As shown in Algorithm 1 lines 4-7, RAG-Sketch adaptively retrieves chunks via the skyline operator defined in Definition 3.5. The operator establishes a partial order among chunks based on two complementary objectives: semantics and presence of keywords. For each query q, we design Algorithm 2 to select the *Pareto optimal chunk set S*, defined as those not

Algorithm 1: RAG-Sketch Pipeline

Input: A large set of external documents $\mathcal{D} = \{d_t\}_{t=1}^{m}$, and a batch of queries $Q = \{q_j\}_{j=1}^{n}$
Output: The derived answers $\{A_j\}_{j=1}^{n}$ for queries Q

1 **foreach** $q_j \in Q$ **do**
2 Decompose query q_j as $\mathcal{M}(q_j) = (E(q_j), d_j, o_j)$;
3 Divide the documents $\mathcal{D}$ into chunks $\{C_i\}_{i=1}^{m}$;
4 **foreach** *query* q_j **do**
5 **foreach** *chunk* C_t *associated with* q_j **do**
6 Compute $\phi(C_t, E(q_j))$, $\omega(C_t, q_j)$;
7 Select Pareto optimal chunks set $S_j \leftarrow$ Algorithm 2;
8 Select final access chunk list and extract target structured data $\mathbb{M}, \mathbb{E} \leftarrow$ Algorithm 3;
9 Derive final answers $\{A_j\}_{j=1}^{n} \leftarrow \text{LLM}(\mathbb{E}, \{q_j\}_{j=1}^{n})$;
10 **return** the answers $\{A_j\}_{j=1}^{n}$

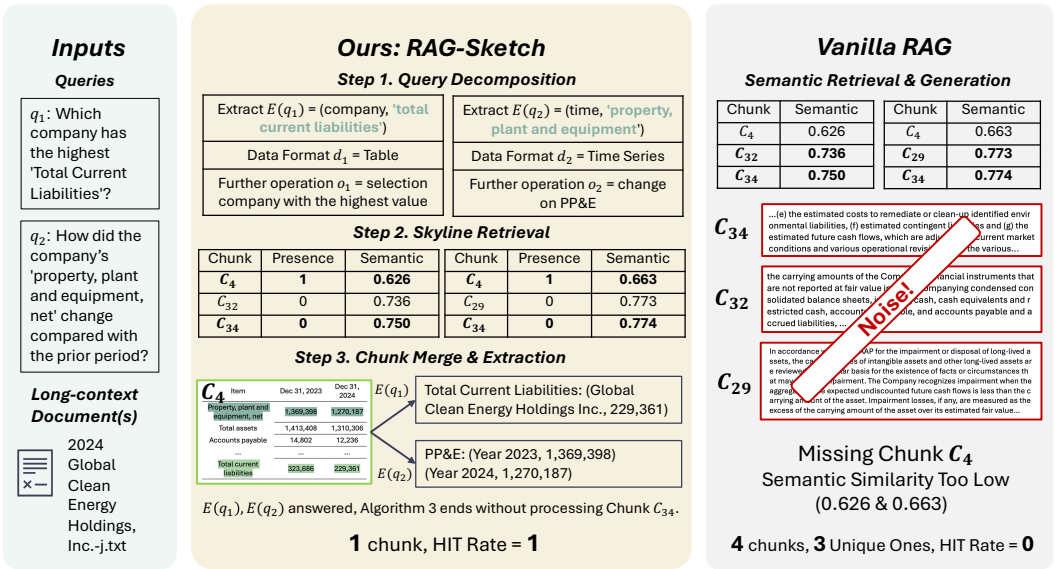


Fig. 3. Running Examples on RAG-Sketch Pipeline in Comparison with Vanilla RAG. All queries and results are from real implementations on Vanilla RAG and RAG-Sketch.

dominated under the operator, thereby ensuring that no chunk with high semantic relevance or strong keyword presence is excluded.

Set Cover Chunk Merging. As shown in Algorithm 1 lines 8-9, we minimize the total input token costs while ensuring that each query q is adequately answered. Specifically, we formalize the chunk merge problem in Problem 3, and prove its NP-hardness in Theorem 3.8. To address this, we perform chunk-to-query mapping to record all meta-queries that map to the same chunk. Then, we propose Algorithm 3 to select merged chunks $\mathbb{M}$ and to extract structured evidence $\mathbb{E}$ with theoretical guarantees.

We explain the pipeline including three main steps with a detailed running example in Figure 3. Through decomposition, skyline retrieval and chunk merge, RAG-Sketch only handles chunk C_4

once to extract the target evidence (HIT Rate = 1). Conversely, Vanilla RAG relies solely on semantic similarity, retrieving $\{C_{32}, C_{34}\}$ for query q_1 and $\{C_{29}, C_{34}\}$ for query q_2. Even with a chunk reuse strategy, Vanilla RAG still requires the processing of three unique chunks and fails to retrieve the gold answer (HIT Rate = 0).

3.2 Query Decomposition

Complex queries in long-context QA often involve multiple steps, making direct retrieval and extraction inefficient or even infeasible. According to the cognitive theories discussed in StructRAG [32], converting raw information into structured forms can improve multi-step query solving. For instance, the query "evaluate the company's recent health in noncurrent assets" is underspecified to directly guide relevant context retrieval and subsequent information extraction. We reformulate it into a structured requirement such as "Extract (company, noncurrent asset)" and specify a target format (e.g., a key–value pair table format), which makes the retrieval target explicit (via the keyword *noncurrent asset*) and provides a clear prompt to guide extraction.

In conclusion, we formalize query decomposition as a structured prediction problem rather than a free-form generation task through LLMs. This is motivated by empirical studies that constraining model outputs to a set of predefined schemas (instead of open-ended generation) improves accuracy and consistency [39, 46, 62].

Additionally, appropriate information representations (IR) for managing extracted information greatly facilitates subsequent reasoning. For example, when ranking companies based on their assets, it is more effective to organize the asset information in a tabular format rather than in a graph. Based on the common downstream operators (comparison/ranking/aggregation/chain/clustering), we adopt a minimal evidence IR: field/value facts (table), relation facts (graph triples), and time-indexed facts (time series), with raw text as a fallback to preserve coverage. The formal definition on query decomposition is as follows.

Definition 3.1 (Query Decomposition). Let Q denote the query space and $\mathcal{M}$ as the space of decomposed queries. A *decomposed query* is defined as a tuple

$$\mathcal{M}(q) = (E(q), d, o),$$

where $E(q)$ is an *extraction meta-query*, d is a target extraction *data structure type*, and o is an optional *operation specification*.

Extraction meta-query $E(q)$ is instantiated under one of four predefined extraction schemas {(key, attribute), (head, relation, tail), (timestamp, value), raw text}, which correspond to structured extraction needs for table, graph, time series, and a text fallback, respectively. Data structure d corresponds to table, graph, time series, and text. Operation o is the further task-specific implementation such as *ranking*, *graph chain construction*, *clustering*, *comparison* or *conditional selection*. For example, "Compare the debt-to-equity ratios of companies in 2023" is decomposed into $E(q)$ that extracts (company, debt-to-equity ratio), d = table and o = comparison values in year 2023. This helps in two ways: (i) $E(q)$ provides a concrete coverage target (e.g., "debt-to-equity ratio") for ϕ during retrieval, and (ii) $E(q)$ and d explicitly guide an intermediate table extraction with the corresponding key-value pairs, which reduces ambiguity for executing o.

Formally, the decomposition function is $f : Q \to \mathcal{M}, \quad q \mapsto (E(q), d, o)$, where f is instantiated by an LLM guided with structured prompts with two-shot examples.

Compared with template-based or chain-of-thought prompting [61], our decomposition into pre-defined schema has higher reliability. Moreover, the inclusion of three structured extraction types plus a text fallback enables the system to handle both structured queries and open-domain general queries in a unified manner.

3.3 Skyline Retrieval

In this subsection, we propose a skyline retrieval strategy that ensures more comprehensive retrieval of scattered information compared to conventional dense/sparse retrieval through dual-dimensional objectives on the decomposed extraction meta-query while reducing token costs by filtering out clearly inferior chunks.

3.3.1 Theoretical Analysis of Human-Inspired Dual-Dimensional Retrieval. Using the same LLM backbone, the primary factor influencing QA accuracy is the comprehensiveness of relevant information retrieval. To achieve this goal without relying on costly context traversal, we draw inspiration from how humans process long-context documents. They typically locate evidence through two main strategies: (1) identifying *explicit mentions* of key entities or attributes (e.g., searching for the keyword "noncurrent assets" in a financial report), and (2) evaluating the *semantic likelihood* that a section or abstract contains relevant information even if the exact keyword is absent. These observations motivate us to formalize two retrieval dimensions that mimic human reasoning.

Let $\{C_1, \ldots, C_n\}$ denote the divided document chunks and the input query q with $E(q)$ as its decomposed extraction meta-query. Each chunk C_i is associated with two signals: a binary presence indicator and a semantic relevance score.

Definition 3.2 (Presence Dimension). The *presence dimension* denoted as $\phi(C_i, E(q)) \in \{0, 1\}$ measures whether a key element, such as an attribute or relation to be extracted given by $E(q)$, appears in chunk C_i, analogous to how humans scan for keywords.

Definition 3.3 (Semantic Dimension). The *semantic dimension*, denoted $\omega(C_i, q) \in [0, 1]$, quantifies the likelihood that C_i contains information relevant to the original input query q.

Score Plateau. To retain multiple highly relevant chunks within the top tier, rather than discarding some due to minor differences in similarity scores, we introduce a score plateau in the semantic dimension. The presence dimension is defined as a binary indicator: $\phi = 1$ if and only if the chunk explicitly contains the required entity or attribute keywords, and $\phi = 0$ otherwise. This binary presence signal provides a direct and reproducible signal, avoiding the issue of discarding chunks based on minor similarity variations. The semantic dimension corresponds to the human strategy of judging contextual relevance from titles, abstracts, or overall semantics with the original query q. For structured documents, ω is estimated based on the chunk abstract or title by querying an LLM about whether the current chunk is likely to contain the answer. If so, ω is assigned a value of 1. For unstructured text, we use semantic similarity to score chunks, then sort scores and dynamically choose the number of retrieved chunks at the elbow (knee) of the score curve to early-stop [52] or choose Top-K ones.

Lemma 3.4 (Complementarity of Dimensions). *Presence and semantic dimensions capture orthogonal aspects of retrieval. In particular, for any chunk C_i and query q,*

$$\phi(C_i, E(q)) = 0 \;\not\Rightarrow\; \omega(C_i, q) = 0.$$

That is, we can still judge a section to be relevant (high ω) even if no explicit keyword appears (low ϕ).

Proof. Because the number of keywords within $E(q)$ is finite, we can form a paraphrase $C^\star$ of q in which each $m \in E(q)$ is replaced by a synonymous expression not contained in $E(q)$. Then no element of $E(q)$ appears in $C^\star$, so by Definition 3.2 we have $\phi(C^\star, E(q)) = 0$. Yet $C^\star$ is semantically equivalent (or highly aligned) with q, hence by the assumed property of ω we obtain $\omega(C^\star, q) > 0$. Taking $C_i = C^\star$ yields the claim. □

Algorithm 2: Pareto Optimal Chunks Retrieval

Input: Chunk list $C = \{C_1, C_2, \ldots, C_n\}$, with pre-calculated values ϕ and ω for each chunk
Output: Pareto optimal set S

```
S ← ∅;
foreach C_i ∈ C do
    if ∃ C_j ∈ S s.t. C_j ≻ C_i then
        continue
    S ← S \ {C_j ∈ S | C_i ≻ C_j}
    S ← S ∪ {C_i}
return S
```

We then provide the formal problem definition on retrieval, based on the preceding theoretical analysis. Specifically, we retrieve chunks that contain the keyword while also ensuring that *no* chunks with high semantic relevance to the given query are missed.

Problem 2 (Adaptive Chunk Retrieval). *Given a list of divided chunks $\{C_1, \ldots C_n\}$, the decomposed extraction meta-query $E(q)$, and the pre-calculated dual dimensions $\phi(C_i, E(q)), \omega(C_i, q)$ for each chunk, a retrieved chunk should be at least as good as any other selected chunk in both dimensions, or better in at least one.*

This formalization highlights that adaptive retrieval should jointly consider both human-inspired objectives: keyword presence and semantic relevance. We defer to later sections how skyline retrieval can naturally resolve the trade-offs implied in Problem 2.

3.3.2 Skyline Operator. To address Problem 2, which requires that selected chunks be at least as good as any other selected chunk in all dimensions or better in at least one, we adapt the skyline operator from traditional database systems. This enables us to solve the retrieval problem effectively, as theoretically analyzed above.

To define the skyline operator, we first formally introduce the **partial order** that underlies it. For dual-dimensional inputs, this partial order is induced by the dominance relationship between chunks, as specified in Definition 3.5. Under this dominance relation, the skyline operator identifies the set of chunks that are *not dominated* by any other, corresponding to the minimal elements in the induced partial order.

Definition 3.5 (Partial Order). For any two chunks C_i, C_j, if:

(1) C_i is not worse than C_j on the Presence dimension as $\phi(C_i, q) \geq \phi(C_j, q)$.
(2) C_i is not worse than C_j on the Semantic dimension as $\omega(C_i, q) \geq \omega(C_j, q)$.
(3) There is at least one dimension on which C_i is strictly better than C_j, as $\phi(C_i, q) > \phi(C_j, q)$ or $\omega(C_i, q) > \omega(C_j, q)$.

Then, we say that C_i *dominates* C_j, denoted as $C_i \succ C_j$.

Based on the partial order, we define the chunks that are *not dominated* by any other on the dual-objectives as the **Pareto optimal chunks** which we intend to select.

Definition 3.6 (Pareto Optimal Chunk Set). Pareto optimal Chunk set S is a set of chunks which are not dominated by others on the defined partial order as shown in Equation (1).

$$S = \{C_i \mid \nexists\, C_j \in S, C_j \neq C_i, C_j \succ C_i\} \tag{1}$$

We then design a Pareto Optimal chunks retrieval algorithm in Algorithm 2. Specifically, we initialize the Pareto optimal chunk set we aim to retrieve as S and iterate over the chunk list to determine whether the current chunk is dominated by any existing chunk in S, based on the Partial Order as illustrated in Definition 3.5.

Specifically, we explain the intuition to address the adaptive chunks retrieval problem through the skyline operator, which is designed to satisfy the following requirements.

- The partial order in Definition 3.5 between chunks defines a dominance relation over the dual objectives. In this way, we can mathematically measure the relationship between two chunks, which facilitates the retrieval of dominant chunks.
- The Pareto optimal chunk definition and retrieval, described in Definition 3.6 and Algorithm 2, identify chunks that dominate others with respect to both objectives. This satisfies the condition in Problem 2, which requires that a selected chunk be at least as good as any other in both dimensions, or better in at least one.
- Score plateau ensures that multiple chunks can share the same semantic score and are all retained by the skyline operator, rather than discarding some due to minor differences in similarity scores. For instance, $\{C_1, C_4, C_6, C_8\}$ are selected in the Table 2 given presence and semantic dimensions.

Table 2. A Toy Example on Skyline Retrieval.

	C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8
Presence (ϕ)	1	1	0	0	0	0	1	0
Semantic (ω)	0.8	0.6	0.7	1	0.6	1	0.5	1

3.4 Set Cover Chunk Merging

In this subsection, we aim to address the high token costs challenge through reducing the chunk access frequency while still ensuring coverage of all queries to be answered.

3.4.1 *Theoretical Analysis of Chunk Merging Problem.*

Example 3.7. Consider the selected Pareto optimal chunks for each meta-query as $S(E(q_1)) = \{C_4, C_5, C_{15}\}$, $S(E(q_2)) = \{C_5, C_6, C_{11}\}$, $S(E(q_3)) = \{C_4, C_{15}, C_{17}\}$, $S(E(q_4)) = \{C_{11}, C_{19}\}$. The corresponding costs for each chunk are: $T(C_4) = 200$, $T(C_5) = 300$, $T(C_6) = 400$, $T(C_{11}) = 450$, $T(C_{15}) = 200$, $T(C_{17}) = 300$, $T(C_{19}) = 200$.

When processing the queries in the traditional rigid query-by-query manner, the total token cost is $(200 + 300 + 200) + (300 + 400 + 450) + (200 + 200 + 300) + (450 + 200) = 3200$. Such approach can result in notably high costs especially in the context of long-input scenarios. To address the high token cost issue, we then give the meta-query merge problem definition as follows.

Problem 3 (Meta-query Merge). *Given the meta-query list $\{E(q_1), E(q_2), \dots\}$, each associated with a Pareto optimal chunk set $S(E(q_j)) = \{C_{j1}, C_{j2}, \dots\}$ and token costs $\{T(C_{j1}), T(C_{j2}), \dots\}$ for each chunk, we aim to construct a final chunk list $\mathbb{M}$ for access, ensuring that the relevant chunk for each query is included to support QA. The objective is to minimize the total cost of accessing chunks, as defined in Equation* (2).

$$F = \min_{\mathbb{M} \subseteq \bigcup_j S(E(q_j))} \sum_{C \in \mathbb{M}} T(C) \tag{2}$$
$$s.t. \quad \forall j,\ \exists\, C \in \mathbb{M},\ C \in S(E(q_j))$$

As shown in Example 3.7, C_5 is shared by both $E(q_1)$ and $E(q_2)$, and can thus be accessed once to serve both queries. We first prove that this problem is NP-Hard and then propose a greedy algorithm to address it.

Theorem 3.8 (Problem 3 is NP-Hard). *The meta-query merge problem is NP-hard via a polynomial-time, cost-preserving reduction from Weighted Set Cover (WSC) [9, 17].*

Proof. **Weighted Set Cover (WSC).** Given a universe U and a family $\mathcal{S} = \{S_1, \ldots, S_m\}$ with weights $w_i \geq 0$, the goal is to pick $\mathcal{S}' \subseteq \mathcal{S}$ covering U with minimum total weight:

$$\min_{\mathcal{S}' \subseteq \mathcal{S}} \sum_{S_i \in \mathcal{S}'} w_i \quad \text{s.t.} \quad \bigcup_{S_i \in \mathcal{S}'} S_i = U. \tag{3}$$

Reduction. From $I_{\text{SC}} = \langle U, \mathcal{S}, \mathbf{w} \rangle$, we build an instance $I_{\text{Merge}} = \langle \{E(q_j)\}, \bigcup_j S(E(q_j)), \mathbf{T} \rangle$ of **Problem 3** as follows. For each meta-query $E(q_j)$, the Pareto optimal chunk list $S(E(q_j))$ is selected by Algorithm 2. Treat each chunk C as a subset of meta-queries: it *covers* all $E(q)$ with $C \in S(q)$. Set its cost $T(C)$ to match the corresponding subset weight w_i. Then, the objective in Equation (5) is exactly the WSC objective with the coverage constraint that each $E(q_j)$ is covered by selected chunk.

Equivalence. (i) *Forward.* Given any feasible set cover $\mathcal{S}^\star \subseteq \mathcal{S}$, map each $S_i \in \mathcal{S}^\star$ to a chunk $\chi(S_i)$ (induced by Algorithm 2) and let $\mathbb{M}^\star = \{\chi(S_i) \mid S_i \in \mathcal{S}^\star\}$. Since $\mathcal{S}^\star$ covers U, $\mathbb{M}^\star$ covers all meta-queries; moreover,

$$\sum_{C \in \mathbb{M}^\star} T(C) = \sum_{S_i \in \mathcal{S}^\star} w_i.$$

(ii) *Backward.* Given any feasible merge set $\mathbb{M}^\star \subseteq \bigcup_j S(E(q_j))$, map each $C \in \mathbb{M}^\star$ to a subset $\phi(C) \in \mathcal{S}$ and set $\mathcal{S}^\star = \{\phi(C) \mid C \in \mathbb{M}^\star\}$. Coverage and cost are preserved:

$$\bigcup_{S_i \in \mathcal{S}^\star} S_i = U, \qquad \sum_{S_i \in \mathcal{S}^\star} w_i = \sum_{C \in \mathbb{M}^\star} T(C).$$

Both χ and ϕ are computable in polynomial time.

In conclusion, WSC is NP-complete; hence this polynomial-time, cost-preserving reduction implies **Problem 3** is NP-hard. □

3.4.2 Chunk-to-Query Mapping Greedy Algorithm. To address Problem 3, we first analyze its underlying requirements. Specifically, the main goal is to minimize the total cost of chunk access, denoted as F as defined in Equation (2). At the same time, we must still satisfy the primary long-context LLM QA task described in Problem 1, which requires that each query be successfully tackled and answered.

We first perform a mapping from each chunk to its associated queries. In other words, for each chunk, we identify all meta-queries that correspond to it, as defined in Equation (4).

$$Q(C) = \{E(q_j) \mid C \in S(E(q_j)), q_j \in Q\} \tag{4}$$

The objective of Problem 3 as defined in Equation (2) can be reformulated in Equation (5) as follows:

$$\min \sum_{C_j \in \mathbb{M}} T(C_j), \quad \text{s.t.} \bigcup_{C_j \in \mathbb{M}} Q(C_j) = Q \tag{5}$$

$\mathbb{M}$ denotes the union of the finally selected chunks to be accessed, while Q represents the set of all queries. We use M to record the covered queries in each iteration of our algorithm, which is

initially set to $\emptyset$. To greedily select chunks that satisfy both constraints, we design an objective function G that incorporates them as follows:

$$G(C_j) = \frac{|\boldsymbol{Q}(C_j) \cap (Q - M)|}{T(C_j)} \tag{6}$$

When selecting a new chunk C_j, we aim to maximize the number of covered meta-queries, given by $|\boldsymbol{Q}(C_j) \cap (Q - M)|$, while minimizing the cost of including C_j, represented by $T(C_j)$. Following this, at each iteration, the algorithm greedily selects the chunk with the highest value of G in Equation (6), thereby satisfying both targeted objectives. The algorithm terminates when all queries have been addressed or when no retrieved chunks remain. Specifically, as shown in Algorithm 3, lines 10–16, for each greedily selected chunk C^* at each iteration, we perform LLM-based structured data extraction on its associated queries $\boldsymbol{Q}(C^*)$. After extracting structured data A_q and appending it to the extracted set $\mathbb{E}_q$, we use a local LLM (LLama2-7b) to determine whether the collected information is sufficient for answering the original query q. If the model confirms sufficiency, we mark that the query q is answered. We then add the corresponding query index to the set of covered meta-queries M.

Algorithm 3: Chunk-to-Query Mapping Greedy Algorithm

Input: The decomposed query list $\{E(q)\}$;
The selected skyline chunk list $E(q_j)$: $S_j = \{C_{j1}, C_{j2}, \dots\}$;
Their corresponding access costs: $\{T(C_{j1}), T(C_{j2}), \dots\}$
Output: The selected chunk list $\mathbb{M}$, extracted data $\mathbb{E}$

1 Construct the chunk-query mapping $\boldsymbol{Q}(C) \leftarrow$ Equation (4);
2 Initialize $L \leftarrow \{C_j \mid C_j \in S(q), E(q) \in Q\}$,;
3 $M \leftarrow \emptyset, \mathbb{M} \leftarrow \emptyset$;
4 **while** $M \neq Q$ ***and*** $L \neq \emptyset$ **do**
5 Select chunk $C^* \leftarrow \arg\max_{C_j \in L} G(C_j)$,;
6 where $G(C_j)$ is computed by Equation (6) with $\boldsymbol{Q}(C_j) \cap (Q - M)$;
7 **if** $\boldsymbol{Q}(C^*) \cap (Q - M) = \emptyset$ **then**
8 **break**;
9 $\mathbb{M} \leftarrow \mathbb{M} \cup \{C^*\}, \quad L \leftarrow L \setminus \{C^*\}, \quad \mathbb{Q} \leftarrow \emptyset$;
10 **foreach** $q \in \boldsymbol{Q}(C^*) \cap (Q - M)$ **do**
11 $A_q \leftarrow \text{LLM}(C^*, E(q))$;
12 **if** A_q *is not void* **then**
13 $\mathbb{E}_q \leftarrow \mathbb{E}_q \cup A_q$;
14 **if** $\mathbb{E}_q$ *is enough for* q **then**
15 $\mathbb{Q} \leftarrow \mathbb{Q} \cup q$;
16 Update $M \leftarrow M \cup \mathbb{Q}$
17 **return** $\mathbb{M}, \mathbb{E}$

We then prove that Algorithm 3 has an approximation ratio compared to the optimal solution.

Theorem 3.9 (Harmonic Approximation Ratio with Probabilistic Answers). *Let $n = |Q|$, let OPT denote the minimum total token cost of any cover, and let $p_{\min} \in (0, 1]$ be a lower bound such*

that, given the extracted evidence up to the current chunk, the LLM answers the query with probability at least $p_{\min}$. Then Greedy Algorithm 3 returns a chunk set with

$$\mathbb{E}[ALG] \le \frac{H(n)}{p_{\min}}\, OPT, \qquad H(n) = \sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n),$$

so the expected approximation ratio is $O\big(\frac{\log n}{p_{\min}}\big)$.

Proof. Let $R_t = Q \setminus M_t$ be the residual set after t iterations, $r_t = |R_t|$, and d_{t+1} as the number of newly covered queries during that step as follows.

$$d_{t+1} = |\mathcal{Q}(C_{t+1}) \cap R_t| \tag{†}$$

We denote B_{t+1} as the number of queries *successfully answered* by LLMs where the queries should be included in M_{t+1}.

$$B_{t+1} \;=\; \sum_{q \in \mathcal{Q}(C_{t+1}) \cap R_t} \mathbf{1}\{\text{success on } (C_{t+1}, q)\}$$

By independence and the lower bound $p_{\min}$, $\mathbb{E}[B_{t+1}] \ge p_{\min} d_{t+1}$. Let the gain-per-cost be

$$G_{t+1}(C) = \frac{|\mathcal{Q}(C) \cap R_t|}{T(C)}.$$

Lemma 3.10. *(Efficiency) For every t, $G_{t+1}(C_{t+1}) \ge \frac{r_t}{OPT}$.*

Proof. Let $O = \{C_1^\star, \ldots, C_k^\star\}$ be an optimal cover with $\sum_i T(C_i^\star) = \mathrm{OPT}$. Since $\bigcup_i \mathcal{Q}(C_i^\star)$ covers R_t, we have $\sum_i |\mathcal{Q}(C_i^\star) \cap R_t| = r_t$, so by averaging some $C^\star$ satisfies $G_{t+1}(C^\star) \ge r_t/\mathrm{OPT}$. □

Charging argument. We charge $T(C_{t+1})$ uniformly to the *successful* answers in step $t+1$. The expected charge per newly answered query is

$$\mathbb{E}\Big[\tfrac{T(C_{t+1})}{B_{t+1}}\Big] \le \frac{T(C_{t+1})}{p_{\min} d_{t+1}} = \frac{1}{p_{\min}} \cdot \frac{1}{G_{t+1}(C_{t+1})} \le \frac{\mathrm{OPT}}{p_{\min}} \cdot \frac{1}{r_t}.$$

Summation. Let $r(q)$ be $|R_t|$ immediately before q is *successfully* covered. Summing the expected charge over all $q \in Q$ gives

$$\mathbb{E}[\mathrm{ALG}] \le \frac{\mathrm{OPT}}{p_{\min}} \sum_{q \in Q} \frac{1}{r(q)} = \frac{\mathrm{OPT}}{p_{\min}} H(n).$$

Upper Bound Truncation (Worst Case). Considering the upper bound above may explode when $p_{\min}$ becomes very small, we give the further proof on the worst case of Algorithm 3 as the truncation. Since Algorithm 3 updates $L \leftarrow L \setminus \{C^*\}$ and never inserts new chunks, each input unique chunk is thus processed at most once. Therefore,

$$\mathbb{E}[\mathrm{ALG}] \le \min\Big\{\tfrac{H(n)}{p_{\min}}\mathrm{OPT},\ T_{\mathrm{L}}\Big\}.$$

T_L is the tokens of the unique pareto optimal chunks L after mapping shown in Algorithm 3 line 2. □

3.5 Discussion and Scope

We discuss RAG-Sketch's adaptability to multimodal scenarios and reflect on the strategic design trade-offs of skyline operator.

Generality and Modality Extension. While RAG-Sketch currently targets textual RAG systems, the underlying framework of skyline operator and batch-level chunk merging modules is modality-agnostic. Extending RAG-Sketch to multimodal scenarios requires replacing textual encoders with cross-modal equivalents (e.g., CLIP [47]) for semantic ranking, while leveraging captions or detected entities as surrogates for visual modality as the presence signal. While a full-scale multimodal evaluation remains outside the current scope, RAG-Sketch will try to extend in the future.

Skyline Operator Granularity and Trade-offs. RAG-Sketch uses a dual-objective skyline operator with semantic relevance $\omega(C, q)$ and a *binary* presence signal $\phi(C, E(q))$. This binary design prioritizes robustness and removes the need for threshold tuning. ϕ can be extended to *graded* signals (e.g., BM25-based coverage or learned importance scores) for finer presence sensitivity. However, Figure 6a shows that graded signals can fluctuate and require "score plateau" designs for stability. Thus, our current design balances efficiency and stable performance. The skyline operator can also include more orthogonal objectives, but this typically enlarges the Pareto optimal set and increases the retrieved chunk budget. Our dual-dimension design already achieves a high HIT rate (Section 4.4); when adding more objectives, we recommend a stricter score plateau truncation to control the retrieved set size.

4 Experiments

In this section, we compare the performance of RAG-Sketch against the state-of-the-art methods including standard RAG, long-context LLM, and RAG with structured data. We also conduct ablation studies, latency measurement, token cost analysis, and experiments with various backbones. Specifically, we intend to investigate the following research questions (RQ):

- **RQ1**.How does RAG-Sketch method perform compared to various baselines in terms of both accuracy and cost efficiency?
- **RQ2**. What is the performance of RAG-Sketch in the ablation study with respect to both QA accuracy and cost savings?
- **RQ3**. How does RAG-Sketch perform with respect to the binary presence and score plateau, surrounding context, and # shots in decomposition prompt for chunk retrieval?
- **RQ4**. What is the latency of RAG-Sketch in online serving mode, end-to-end (e2e) mode and combined with prefix-caching?
- **RQ5**. What is the token cost of the baseline when using chunk reuse and caching as well as in the overhead modules?
- **RQ6**. How does RAG-Sketch perform with different backbones?

4.1 Experimental Settings

In this subsection, we describe experimental settings of our work, including the datasets, baseline methods, and the evaluation metrics. All experiments are conducted on a server equipped with 256 GB RAM, an 18-core 2.60 GHz CPU, and running CentOS Linux release 7.9.2009 (Core).

4.1.1 Datasets. During evaluation, we adopt the widely used Loong benchmark for long-context QA [59]. The Loong benchmark includes four distinct tasks across three domains and supports four scaling length settings, effectively simulating real-world long-context scenarios. Specifically, the tasks span four categories: *Spotlight Locating*, *Comparison*, *Clustering*, and *Chain of Reasoning*.

Table 3. Statistics and batching details. Avg # Retrieved Chunks denotes the mean number of retrieved chunks per instance, and Avg Tokens per Chunk denotes the average chunk token length divided from the inputs.

Task	Set	# Test (Batch Size)	Avg Token	Avg # Retrieved Chunks	Avg Token per Chunk
Spotlight	Set 1	53	37.0K	9.70	827.48
	Set 2	70	72.8K	9.09	1086.50
	Set 3	80	138.6K	13.09	1369.94
	Set 4	47	245.4K	29.09	1347.57
Comparison	Set 1	60	38.1K	16.67	981.78
	Set 2	105	77.7K	23.12	1469.19
	Set 3	95	144.3K	31.71	2046.02
	Set 4	40	234.1K	34.74	1678.61
Clustering	Set 1	113	37.7K	12.16	1699.69
	Set 2	246	70.4K	13.89	2915.52
	Set 3	194	129.1K	25.60	2789.60
	Set 4	88	212.8K	26.07	2621.7
Chain	Set 1	97	34.3K	17.97	1340.86
	Set 2	143	68.4K	12.02	1878.64
	Set 3	112	128.2K	17.56	2841.06
	Set 4	57	216.7K	30.16	2198.52

- *Spotlight Locating* task is the simplest, where the evidence is contained within a single document among multiple sources.
- *Comparison* task involves comparing multi-source information.
- *Clustering* task evaluates the model's ability to aggregate information across multiple documents.
- *Chain of Reasoning* task requires identifying evidence across multiple documents and establishing relationships among them.

Loong benchmark spans three domains: financial reports, legal cases, and academic papers. It includes 574 financial reports, primarily comprising the most recent quarterly and annual filings from both English and Chinese companies. The legal domain consists of 629 Chinese legal documents, exclusively sourced from cases adjudicated by higher and intermediate courts. For the academic domain, 764 of the latest research articles are collected from arXiv. Table 3 provides detailed data statistics and batching configurations used in our evaluation, including batch size, average token, the average # of retrieved chunks, and the total tokens per batch.

4.1.2 Baselines. We compare our RAG-Sketch with several competitive baselines evaluated in the benchmark. These baselines are broadly categorized into three types: RAG and its optimization, long-context LLM, RQ-RAG and GraphRAG. Specifically, standard RAG (2020) [29] retrieves the Top-K most relevant chunks and passes them to LLMs for answer generation. In our evaluations, the RAG approach uses the OpenAI Embedding Model [43] for retrieval. We employ various values of $K \in \{10, 30, 50\}$. The chunk size is fixed at 1024 tokens to align with the Loong benchmark settings [59]. Long-context LLM (2024) [59] introduces extrapolation techniques to extend the input window up to 128K tokens. RQ-RAG (2024) [10] employs a trained LLM to decompose and refine the original question in order to more accurately identify the required chunk augmentation. GraphRAG (2024) [14] iterates through documents to extract knowledge for subsequent reasoning.

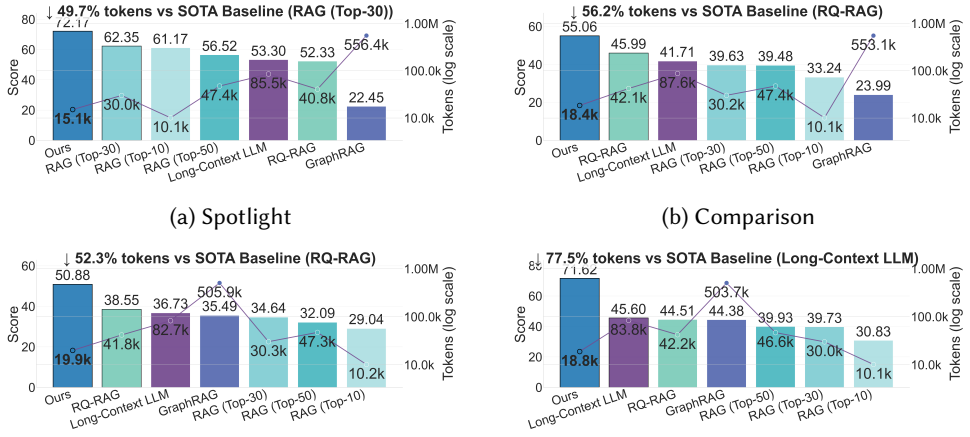
4.1.3 Evaluation Metrics. We adopt three evaluation metrics focusing on both prediction accuracy and token costs. The prediction accuracy is measured using two benchmark-aligned metrics: the average score and the Exact Match (EM) rate. Specifically, we follow the benchmark [59] to leverage GPT-4o [44] as an automatic judge to evaluate generated answers against gold-standard references. The prediction accuracy evaluation considers three key dimensions: accuracy, hallucination, and

Table 4. Performance comparison on LLM-judged scores, exact match rate, and token costs. From Set 1 to Set 4, task complexity gradually increases, as reflected by the growing input tokens. The best result is highlighted in **bold**, while the second-best is highlighted with underline. Ours achieves the highest average prediction accuracy while maintaining low token consumption.

Method	Spotlight			Comparison			Clustering			Chain of Reasoning		
	Score	EM	Tokens	Score	EM	Tokens	Score	EM	Tokens	Score	EM	Tokens
Set 1 (10-50K)												
Long-Context LLM	68.49	0.55	37018.79	60.60	**0.37**	38077.03	47.08	0.08	37714.16	70.39	0.36	34338.15
RAG (Top-50)	51.08	0.35	37014.42	44.53	0.27	36698.77	37.96	0.05	37227.17	53.95	0.35	34340.06
RAG (Top-30)	57.26	0.40	28563.49	45.43	0.28	29159.05	40.04	0.06	29528.38	57.32	0.35	28299.91
RAG (Top-10)	59.81	0.43	9988.57	34.93	0.15	9954.87	29.33	0.02	10088.48	41.27	0.15	10101.71
RQ-RAG	72.31	0.54	35483.56	48.16	0.27	35772.26	47.44	0.07	41965.70	58.96	0.25	40319.63
GraphRAG	31.67	0.00	167327.35	27.60	0.00	171577.73	40.71	0.14	170287.25	54.29	0.43	154348.49
Ours	**79.06**	**0.74**	7003.02	**62.33**	0.25	12264.17	**61.99**	**0.21**	9908.72	**82.94**	**0.58**	21015.79
Set 2 (50-100K)												
Long-Context LLM	64.53	0.43	72820.53	42.60	0.21	74878.43	38.52	0.05	69907.62	51.18	0.20	68393.15
RAG (Top-50)	55.94	0.32	50718.59	47.94	**0.31**	50874.76	34.32	0.03	49732.33	46.64	0.21	49965.77
RAG (Top-30)	66.27	0.46	30370.73	46.28	**0.31**	30514.06	38.95	0.05	30473.85	46.15	0.22	30386.27
RAG (Top-10)	67.07	0.53	10070.17	44.30	0.27	10146.01	34.31	0.05	10182.76	34.03	0.06	10121.80
RQ-RAG	57.35	0.46	42198.73	50.83	0.16	43658.24	42.85	0.03	43187.59	47.60	0.10	42469.25
GraphRAG	24.80	0.00	328563.39	14.29	0.00	337098.74	37.86	0.00	314585.30	46.25	0.12	307853.55
Ours	**76.79**	**0.61**	10348.06	**58.62**	**0.31**	15001.64	**55.46**	**0.06**	11056.57	**71.71**	**0.31**	11004.97
Set 3 (100-200K)												
Long-Context LLM	46.99	0.27	115727.00	37.06	0.13	116815.25	31.50	0.02	105622.02	35.01	0.07	111197.83
RAG (Top-50)	67.44	0.50	50985.20	41.82	0.24	51055.93	31.59	**0.04**	50997.80	37.29	0.12	50923.02
RAG (Top-30)	**73.69**	**0.55**	30553.81	42.20	**0.27**	30611.78	32.78	0.02	30588.28	37.65	0.13	30563.63
RAG (Top-10)	67.50	0.46	10160.42	33.44	0.16	10194.79	27.94	0.02	10189.18	31.62	0.06	10170.94
RQ-RAG	50.50	0.13	43572.33	44.62	0.00	46387.81	36.98	0.00	41653.56	36.79	0.07	44287.08
GraphRAG	15.83	0.00	624393.42	27.40	0.00	649905.85	42.50	0.00	581037.49	43.33	0.17	576976.96
Ours	69.31	0.53	14001.15	**54.95**	0.19	20725.19	**46.19**	**0.04**	20731.12	**67.59**	**0.29**	16643.80
Set 4 (200-250K)												
Long-Context LLM	33.18	0.16	116514.89	26.59	0.08	120457.08	29.84	**0.01**	117699.20	25.81	0.04	121086.84
RAG (Top-50)	51.63	0.26	51070.19	23.62	0.08	51125.80	24.49	0.00	51093.24	21.84	0.02	51061.42
RAG (Top-30)	52.17	0.24	30619.98	24.60	**0.10**	30674.05	26.78	0.00	30649.44	17.79	0.00	30623.82
RAG (Top-10)	50.32	0.28	10162.85	20.30	0.03	10216.55	24.56	0.00	10228.36	16.38	0.00	10165.05
RQ-RAG	29.17	0.08	41765.97	40.36	0.00	42517.51	26.92	0.00	40428.21	34.69	0.00	41693.21
GraphRAG	17.50	0.00	1105298.78	26.67	0.00	1053683.15	20.91	0.00	957721.37	33.67	0.33	975783.92
Ours	**63.51**	**0.36**	29072.83	**44.35**	0.03	25794.45	**39.89**	0.00	38070.64	**64.23**	**0.35**	26578.96

completeness, with each answer receiving a score ranging from 0 to 100. Regarding cost evaluation, we follow the definition provided in Definition 2.4. While the overall cost of LLM usage depends on both input and output token lengths [30], the output length is generally uncontrollable. Additionally, in the long-context QA scenario, the input length, which spans from tens of thousands to hundreds of thousands of tokens, is significantly larger than the output length and thus becomes the dominant factor in cost. The detailed three metrics are as follows.

- The **average score** denoted as Score is calculated as the mean score across all instances.
- **Exact Matching (EM)** denotes the proportion of responses that achieve a perfect score (100).
- The **token cost** is the input number of tokens.



(a) Spotlight (b) Comparison

(c) Clustering (d) Chain of Reasoning

Fig. 4. Score (left-axis bars) and Token (right-axis curve) comparison. All methods are ranked by Score.

4.2 Overall Comparison (RQ1)

We compare RAG-Sketch with various baselines on the Loong benchmark, which consists of four tasks across four input length scales. As shown in Table 4, we evaluate RAG-Sketch against long-context LLM, RAG (Top-K) with the varying number of retrieved chunks, RQ-RAG, and GraphRAG. To ensure a fair comparison, all methods, including RAG-Sketch, use the same backbone aligned with the benchmark [59], specifically Qwen2.5-72B [64]. Furthermore, evaluations are conducted using three metrics to assess both prediction accuracy and token costs. Overall, RAG-Sketch achieves the best performance in terms of both Score and EM. This performance advantage can be attributed to the skyline retrieval module combined with query decomposition, which ensures comprehensive coverage of scattered information. These two components explicitly show the information extraction requirements and enable the retrieval of all chunks that are likely to contain the answers. In terms of token costs, RAG-Sketch maintains competitive efficiency, enabled by both the skyline retrieval module and the chunk merging module. Based on the experimental results presented in Table 4, we provide detailed observations and analysis as follows.

- RAG-Sketch achieves the highest average Score and EM on average across all four tasks while maintaining low token costs when compared to other baselines in the benchmark. We provide per-task plots that visualize token costs using a line plot on a log-scale alongside score bars, and rank all methods in each subplot by score (high → low) for easier comparison as shown in Figure 4. There is token reduction of our method against the state-of-the-art baseline (SOTA) in each task (e.g., 77.5% fewer tokens than SOTA in Chain of Reasoning task), highlighting RAG-Sketch's reduced token costs of LLM API calling while maintaining superior scores. This performance advantage can be attributed to the effectiveness of the query decomposition module and the skyline retrieval algorithm, which explicitly guide the information extraction and enable the retrieval of the most relevant chunks.
- RAG-Sketch maintains a comparable token cost between RAG (Top-10) and RAG (Top-30), which retrieve the top 10 and top 30 semantically similar chunks, respectively. Despite similar cost levels, RAG-Sketch consistently outperforms RAG (Top-10) and RAG (Top-30) in prediction accuracy, with average improvements of 72.8% and 45.5% across all tasks and data scales, respectively.

In addition, RAG-Sketch achieves 192.7% and 68.5% higher EM across these tasks on average, significantly surpassing the results of RAG (Top-10) and RAG (Top-30).

- GraphRAG is effective only when the target evidence can be naturally organized as a KG. In the Loong benchmark, such KG-friendly queries mainly appear in the Clustering and Chain of Reasoning tasks. This is reflected in Figure 4, where GraphRAG performs relatively stronger on these two tasks. GraphRAG is among the top baselines on Clustering (35.49) and on Chain of Reasoning (44.38). However, RAG-Sketch decomposes the query and selects an appropriate extraction format (e.g., tables, time series, or KGs), leading to better overall performance.

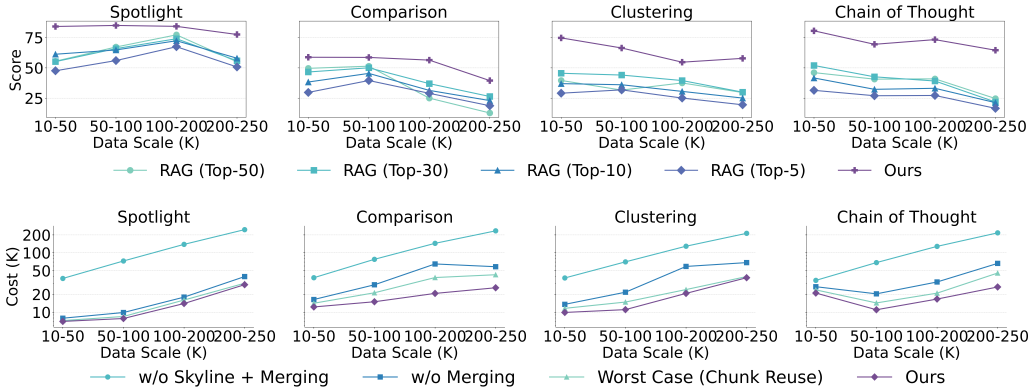


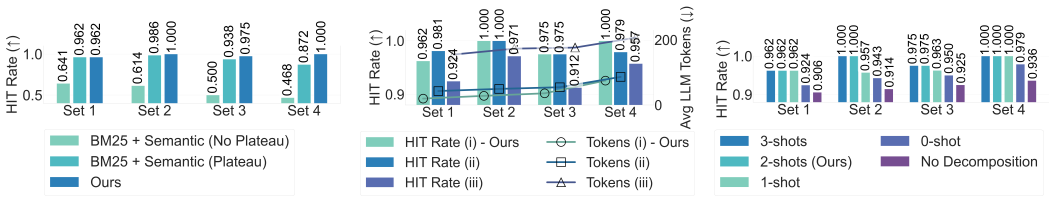
Fig. 5. Ablation Study on QA Accuracy via *Score* (higher is better) and on token costs via *Cost* in log scale (lower is better).

4.3 Ablation Study (RQ2)

For the long-context QA task, we evaluate performance in terms of both prediction accuracy and token costs. As discussed in Section 3, the adaptive skyline retrieval based on the decomposed extraction meta-query, which controls the quality of retrieved evidence, is the only component in RAG-Sketch that affects QA accuracy. In contrast, both modules of RAG-Sketch contribute to token cost savings, as adaptive skyline retrieval and set-cover chunk merging jointly reduce the number of tokens processed. Thus, for the ablation study on **QA accuracy**, we replace the first module with the standard dense retrieval strategy in traditional RAG. We evaluate four different values of K, as shown in Figure 5. Specifically, RAG-Sketch (Ours) achieves significantly higher average scores across all K settings and four different tasks in the ablation study.

Secondly, to assess the impact of the proposed modules in RAG-Sketch with respect to **cost savings**, we design two groups of ablation experiments. In the first group, we remove the greedy chunk merging module in Algorithm 3, denoted as "w/o Merging". In the second group, we remove both the merging module and the skyline operator for the adaptive Pareto optimal chunk retrieval in Algorithm 2, denoted as "w/o Skyline + Merging". In the third group, we apply the chunk reuse strategy to the Pareto optimal chunks identified by the skyline retrieval module, denoted as "Worst Case (Chunk Reuse)". Specifically, chunk reuse ensures that each unique chunk in a batch is accessed only once, serving as the worst case for the greedy selection process described in Algorithm 3. As shown in Figure 5, removing either the skyline operator or merging module (denoted as "w/o Skyline + Merging" and "w/o Merging") leads to a significant increase in input token costs. This demonstrates the effectiveness of our proposed modules in reducing retrieval overhead. Specifically,

the skyline operator for Pareto optimal chunk selection module alone reduces costs by 56.0%, 72.0%, 67.8%, and 74.4% across four tasks on Set 1 to Set 4, respectively, when compared with the variant lacking both modules. Moreover, RAG-Sketch further reduces costs by 22.4%, 44.5%, 58.6%, and 48.7% compared to the version without the merging module. On average, we reduce the billed token cost by 23.33% compared to "Worst Case (Chunk Reuse)" curve, due to our greedy algorithm that addresses the reuse process as an optimization problem.



(a) Binary Indicator and Score Plateau

(b) Surrounding Context around Title/Abstract

(c) Decomposition and Prompt Design

Fig. 6. Influence analysis of each module in Skyline Retrieval on Spotlight Task across four data scales.

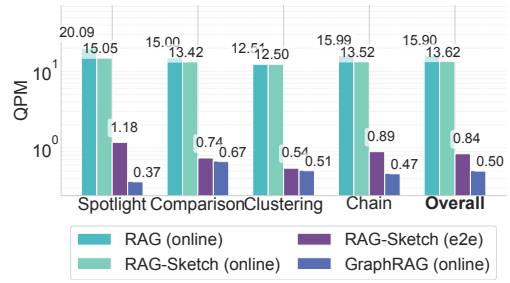
4.4 Analysis of Skyline Retrieval (RQ3)

In this subsection, we analyze the effectiveness of each proposed strategy within the skyline retrieval module (Section 3.3). We use the HIT rate, defined as the accuracy with which retrieved chunks contain the gold answer provided in the Loong benchmark's spotlight task. Our evaluation includes three settings: (1) Figure 6a assesses the binary indicator design (BM25 [21]) using a score plateau or not for the presence dimension. The results indicate that using BM25 without a plateau significantly reduces the HIT rate, whereas incorporating a plateau allows BM25 to perform comparably to our method. (2) Figure 6b evaluates the efficacy of the semantic dimension using q+title/abstract compared to more complex contexts. We compare (i) q+title/abstract (ours), (ii) q+title/abstract + surrounding context (the two subsequent sentences), and (iii) q+summary generated via an LLM. As shown in Figure 6b, (i) achieves robust HIT rates (0.962–1.000) across all sets. In contrast, (ii) does not guarantee consistent improvements, and (iii) consistently degrades the HIT rate, likely because LLM-generated summaries may omit core semantics. (3) Figure 6c examines how query decomposition influences the extraction meta-query $E(q)$, which guides the presence dimension calculation $\phi(C, E(q))$. The results suggest that query decomposition is essential, as all decomposition settings outperform the non-decomposed baseline. Furthermore, a two-shot prompt provides the most cost-effective design; increasing to three shots yields no further gains, while one shot proves insufficient.

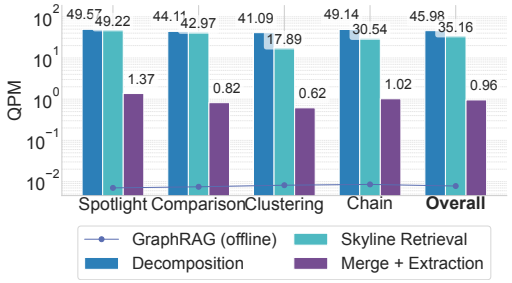
4.5 Latency Evaluation (RQ4)

In this subsection, we investigate practical system latency by evaluating Queries Per Minute (QPM) and Time To First Token (TTFT), as illustrated in Figure 7. Specifically, QPM is derived from wall-clock time using the formula $\text{QPM} = 60 \cdot N/T$, where N denotes the number of test queries and T represents the total runtime. Our evaluation considers three distinct latency configurations: (i) online serving mode, (ii) e2e mode, and (iii) the vLLM-driven [25] implementation of RAG-Sketch.

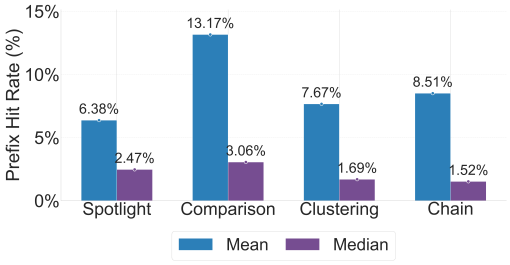
Online serving: RAG-Sketch improves accuracy score and Exact Match over vanilla RAG by +57.25% and +155.05%, respectively, while maintaining comparable throughput (12.55 vs. 14.77 QPM) with vanilla RAG (online), i.e., a sub-second latency difference (4.78s/query vs. 4.06s/query).



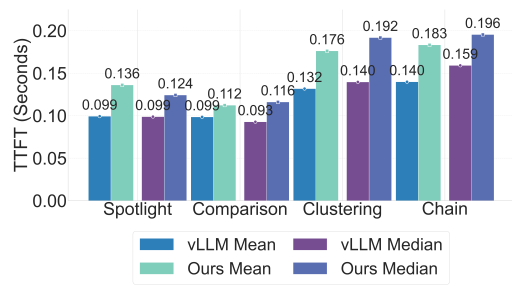
(a) QPM Comparison on Two Modes



(b) RAG-Sketch offline stage breakdown



(c) Prefix Hit Rate



(d) TTFT

Fig. 7. Latency Evaluation. Bars in Figure 7a and Figure 7b show the average QPM. The curve reports GraphRAG's offline throughput (corpus processing per minute). Figure 7c reports the prefix hit rate of vLLM-driven RAG-Sketch during generation phase. Figure 7d reports the corresponding latency improvement on RAG-Sketch driven by vLLM.

Table 5. Two overhead modules invoking tokens

	Query Decomposition Cost		Chunk Retrieval Cost	
Classification	**Prompt**	**Inputs**	**Prompt**	**Inputs**
AVG Tokens	294	215.77	55	800.15

RAG-Sketch (online) also delivers a 26.97× gain over GraphRAG (online) (0.505 QPM). **End-to-end (e2e) execution:** The average e2e throughput for RAG-Sketch is 5.39 QPM (11.13 s/query). Unlike our method, GraphRAG relies on corpus-level KG construction. For an average input of 105.2K tokens, this indexing process takes 2.10 hours (~ 0.0079 corpus/min), demonstrating that GraphRAG's offline cost is substantially higher than our preprocessing overhead. **Prefix-caching (KV reuse)** improves RAG performance by reusing KV states for repeated chunks during generation. It enhances TTFT and saves token costs after feeding into LLMs during generation phase. RAG-Sketch and prefix-caching are complementary. We thus combine RAG-Sketch with vLLM [25] with results shown in Figure 7c and Figure 7d. Results confirm that combining RAG-Sketch with vLLM further achieves a ~8% prefix cache hit rate (reusing ~8% of tokens in the generation prefill) and improves TTFT by ~22%, indicating that prefix caching complements RAG-Sketch to improve cost-effectiveness and latency over RAG systems.

4.6 Tokens Comparison (RQ5)

In this subsection, we evaluate token costs of RAG-Sketch's overhead LLMs invoking, and tokens of baselines using chunk reuse strategy and caching mechanism.

Table 6. Tokens Comparison on Chunk Reuse and Caching

Methods	Spotlight	Comparison	Clustering	Chain
LC-LLM (Origin)	85,520.30	87,556.95	81,110.41	85,379.34
LC-LLM (Chunk Reuse)	69,499.48	41,741.23	31,523.56	48,816.24
GraphRAG (Origin)	556,395.74	553,066.37	505,907.85	503,740.73
GraphRAG (LRU)	537,253.32	313,474.44	364,902.70	406,407.94
GraphRAG (LFU)	549,497.23	491,788.60	497,755.01	451,859.11
Skyline + Chunk Reuse	15,618.25	29,115.80	22,583.17	26,165.41
Ours (RAG-Sketch)	**14,477.99**	**18,446.36**	**19,941.76**	**18,810.88**

We report the average overhead tokens in Table 5. The query decomposition stage uses 294+215.77 tokens on average, and the chunk retrieval stage uses 55 + 800.15 tokens. Compared to the average token cost across all methods for final generation in Table 4, which is 108,843.75, the overhead accounts for only $(294 + 215.77 + 55 + 800.15)/108843.75 = 1.25\%$ per query.

To ensure a fair evaluation when baselines adopt batching or caching to reduce billed tokens, we summarize the results in Table 6. Caching reduces billed tokens only by avoiding repeated LLM invocations. Thus, we implement static-cache variants (LRU/LFU) for GraphRAG preprocessing, which repeatedly calls an LLM to produce intermediate outputs (e.g., knowledge graph construction). LRU and LFU reduce GraphRAG's preprocessing billed tokens by 23.46% and 6.05%, respectively. Nevertheless, RAG-Sketch still achieves substantially lower overall billed token cost: GraphRAG (LRU/LFU) incurs 405,509.60/497,724.99 tokens, whereas ours incurs 18,076.32 tokens (95.54%/96.37% fewer). For batch-level chunk reuse, we apply the same deduplication strategy to the long-context baseline ("LC-LLM (Chunk Reuse)") and to our skyline-retrieved chunks ("Skyline + Chunk Reuse") under the same batching setting (Table 3). Chunk reuse ensures each unique retrieved chunk is included at most once, which serves as a worst case of our merging module (Algorithm 3). Our greedy merging further reduces billed token cost by 23.33% on average relative to this worst case.

4.7 Backbone Influences (RQ6)

In this subsection, we investigate the robustness of RAG-Sketch across various backbone LLMs. Specifically, the backbone models include GPT-4o, GPT-4o-mini, Deepseek-V3, and Qwen2.5-72B-Instruct. As shown in Table 7, RAG-Sketch equipped with various backbones demonstrates robust performance across four tasks and four data scales. Specifically, GPT-4o achieves the highest prediction scores and exact match metrics. However, it also incurs the highest API calling fees per token. Therefore, we recommend DeepSeek-V3, which offers comparable performance with lower API costs. Furthermore, as larger size of the test set ranges from 10K to 250K, the performance gap between set 1 to set 4 is acceptable 18% across different backbone models. Finally, we observe that DeepSeek-V3 and Qwen2.5-72B-Instruct outperform GPT-4o on the Chain of Reasoning task, which emphasizes logical deduction. This is possibly because DeepSeek-V3 and Qwen2.5-72B-Instruct are optimized for structured reasoning and logical deduction.

5 Related Work

In this section, we review related work on enhancing LLM-based QA using a local corpus. The related approaches fall into two main categories. The first directly processes long-context inputs using advanced closed-source LLMs. The second category is RAG and its optimizations, which mainly follows relevant facts retrieval to support answer generation [19, 38, 59].

Table 7. Various LLM Backbones Evaluation on RAG-Sketch Framework.

Backbone	Spotlight		Comparison		Clustering		Chain of Reasoning	
	Score	EM	Score	EM	Score	EM	Score	EM
Set 1 (10-50K)								
DeepSeek-V3	74.43	0.40	64.50	0.28	71.50	0.27	85.15	0.65
Qwen2.5-72B	79.06	0.74	62.33	0.25	61.99	0.21	82.94	0.58
GPT-4o-mini	77.55	0.66	56.75	0.28	54.60	0.12	79.48	0.53
GPT-4o	84.15	0.75	58.75	0.30	74.73	0.36	80.57	0.54
Set 2 (50-100K)								
DeepSeek-V3	79.57	0.54	63.00	0.34	64.13	0.13	76.86	0.50
Qwen2.5-72B	76.79	0.61	58.62	0.31	55.46	0.06	71.71	0.31
GPT-4o-mini	73.79	0.59	54.38	0.30	45.01	0.05	68.25	0.30
GPT-4o	85.00	0.84	58.62	0.30	66.40	0.17	69.44	0.37
Set 3 (100-200K)								
DeepSeek-V3	77.06	0.59	59.63	0.33	53.74	0.06	75.23	0.45
Qwen2.5-72B	69.31	0.53	54.95	0.19	46.19	0.03	67.59	0.29
GPT-4o-mini	70.63	0.56	50.37	0.31	36.26	0.01	61.92	0.24
GPT-4o	84.25	0.71	56.40	0.22	54.69	0.06	73.30	0.39
Set 4 (200-250K)								
DeepSeek-V3	62.45	0.30	45.88	0.20	50.85	0.01	72.54	0.44
Qwen2.5-72B	63.51	0.36	44.35	0.03	39.89	0.00	64.23	0.35
GPT-4o-mini	54.57	0.30	38.25	0.10	35.34	0.00	49.47	0.05
GPT-4o	77.55	0.62	39.50	0.08	57.84	0.00	64.56	0.25

5.1 Long-context Frontier Closed-source LLMs

Frontier closed-source LLMs [44, 64] such as GPT-4, GPT-5.1, as well as Google Gemini 2.5 Pro expose extremely long context windows (400K for GPT-5.1 and 1,048,576 input tokens for Gemini 2.5 Pro) to directly process long-context inputs without truncation. To achieve such enlarged context window, these systems typically couple long-sequence training with serving-side optimizations. For example, prompt or context caching and discounted cached-input usage is introduced to make long prompts feasible and reusable across requests. However, a longer window does not automatically yield effective long-context QA performance. Similar to humans, LLMs tend to forget the middle portions when processing long input texts [16]. To address this issue in QA tasks, some existing methods focus on mitigating it by compressing the inputs [2, 18, 74].

Empirical studies show that given long context inputs for such frontier models, information positioned near the middle of a long prompt is disproportionately forgotten, known as the "lost-in-the-middle" effect [11, 53]. Modern pipelines therefore insert the inputs compression stage with two goals, re-ordering passages and reducing overall length. Early work uses rule-based heuristics metrics and inverse document frequency to prioritize diversity and mitigate redundancy [5], whereas recent systems employ lightweight LLMs as learned cross-encoders that directly predict passage utility [75]. On the other hand, the reducing works reduce the total length by removing redundant tokens via extractive approaches by prompting small language models (SLMs) [37, 48] or abstractive approaches by re-writing individual passages into compact paraphrases [54].

5.2 RAG and its Optimizations

The RAG pipeline primarily consists of two stages: fact retrieval and answer generation. Fact retrieval in existing works typically involves two main steps: indexing and retrieval using the constructed index [16]. Some approaches further introduce a pre-indexing module to process the

user query [36, 72], or refine chunk division to preserve semantic coherence, aiming to improve retrieval performance. In the generation phase, RAG generates answers based on the retrieved facts. To reduce GPU computation costs and latency during this phase, KV cache and related techniques are employed.

Pre-indexing. The pre-indexing module involves document chunking and optimization. To stay within token limits, existing chunking methods often apply overlapping windows [68], preserve sentence boundaries and semantic coherence [15], and adapt to domain-specific structures [55]. In addition, some works perform user query pre-processing to refine the original query for improved retrieval. This includes techniques such as query routing, query rewriting, and query expansion [36, 72], with the goal of producing clearer and more retrieval-suitable queries [16].

Indexing and Retrieval. For standard RAG, the indexing module involves textual representation through either sparse embeddings such as Bag of Words [51] and TF-IDF [50] or dense embeddings such as BM25 [21], GloVE [45], and BERT [13]. Vector representations are then stored in Approximate Nearest Neighbour (ANN) indexes provided by Faiss or other higher-level toolkit [26, 34]. In contrast, RAG with structured data first extracts structured information, such as graphs and tables, by iterating over the documents. It then applies Leiden-based [57] hierarchical community detection so that queries can first hit a global community, then drill into local neighborhoods [14]. Existing retrieval approaches can broadly be divided into two complementary paradigms. The first is semantic similarity ranking [69]. The second paradigm is two-stage semantic re-ranking, which re-scores the top-K retrieved candidates with a more precise but computationally heavier cross-encoder [42, 49].

Generation and Chunk Caching Optimizations. The generation phase is split into two steps, a prefill pass that processes the prompt (including retrieved facts) and an autoregressive decode pass. The KV cache plays a vital role in storing attention states to avoid redundant calculations. Standard KV caching mainly accelerates within-request decoding, while cross-request KV reuse (often called prefix caching) only works when prompts share an identical prefix. Thus, this is brittle for RAG because retrieved chunks and their ordering can vary even under small query changes. To address this, chunk caching as a RAG-oriented subclass of KV-cache reuse can cache precomputed KV states for individual retrieved chunks (chunk caches) and reuses them across requests to reduce prefill compute [24]. A primary example is Cache-Craft [1], which optimizes this approach by identifying reusable chunks regardless of their surrounding context. Instead of a full recompute, it fixes caches by selectively updating only the affected tokens or layers to maintain output quality. This shift toward modularity is echoed in other recent systems like RAGCache [20], which utilizes a multi-level GPU-host hierarchy for document states, and CacheBlend [65] or TurboRAG [35], which focus on the complex task of blending these precomputed caches into a single, coherent prompt.

6 Conclusions

In this paper, we propose RAG-Sketch, a novel and cost-effective framework for long-context QA with LLMs. RAG-Sketch addresses the key challenges of achieving comprehensive information retrieval while minimizing computational costs by introducing multi-choice query decomposition, adaptive skyline chunk retrieval, and an efficient set-cover chunk merging algorithm with theoretical guarantees. Experiments on widely used benchmarks demonstrate that RAG-Sketch consistently outperforms existing methods in both accuracy and cost saving. These results highlight the potential of RAG-Sketch to make the RAG framework more practical and scalable for real-world applications that require processing large and complex document collections.

Acknowledgments

The authors would like to thank the anonymous reviewers for their insightful reviews. Lei Chen is supported by National Key Research and Development Program of China Grant No. 2023YFF0725100, National Science Foundation of China under Grant No. U22B2060, Guangdong-Hong Kong Technology Innovation Joint Funding Scheme Project No. 2024A0505040012, AOE Project AoE/E-603/18, Theme-based project TRS T41-603/20R, CRF Project C2004-21G, Key Areas Special Project of Guangdong Provincial Universities 2024ZDZX1006, Guangdong Province Science and Technology Plan Project 2023A0505030011, HKUST(GZ) CMCC(Guangzhou Branch) Metaverse Joint Innovation Lab under Grant No. P00659, Hong Kong ITC TC-SKLCRCC26EG01, ITF grant PRP/004/22FX, Zhujiang scholar program 2021JC02X170, HKUST Webank joint research lab. Haoyang Li is supported by PolyU P0052504, PolyU-Huawei P0053707, PolyU-Minshang P0057770, PolyU TDG25-28/TBP/11-IICA. Yongqi Zhang's work is supported by Guangdong Basic and Applied Basic Research Foundation 2025A1515010304, Guangdong Province Project 2024QN11X088, Guangzhou Science and Technology Planning Project 2025A03J4491.

References

[1] Shubham Agarwal, Sai Sundaresan, Subrata Mitra, Debabrata Mahapatra, Archit Gupta, Rounak Sharma, Nirmal Joshua Kapu, Tong Yu, and Shiv Saini. 2025. Cache-Craft: Managing Chunk-Caches for Efficient Retrieval-Augmented Generation. *Proc. ACM Manag. Data* 3, 3, Article 136 (June 2025), 28 pages. doi:10.1145/3725273

[2] Nathan Anderson, Caleb Wilson, and Stephen D Richardson. 2022. Lingua: Addressing scenarios for live interpretation and automatic dubbing. In *Proceedings of the 15th Biennial Conference of the Association for Machine Translation in the Americas (Volume 2: Users and Providers Track and Government Track)*. 202–209.

[3] Yihao Ang, Yifan Bao, Qiang Huang, Anthony K. H. Tung, and Zhiyong Huang. 2024. TSGAssist: An Interactive Assistant Harnessing LLMs and RAG for Time Series Generation Recommendations and Benchmarking. *Proc. VLDB Endow.* 17, 12 (2024), 4309–4312. doi:10.14778/3685800.3685862

[4] Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language Models Enable Simple Systems for Generating Structured Views of Heterogeneous Data Lakes. *Proc. VLDB Endow.* 17, 2 (2023), 92–105. doi:10.14778/3626292.3626294

[5] Vladimir Blagojevi. 2023. Enhancing rag pipelines in haystack: Introducing diversityranker and lostinthemiddleranker.

[6] Jan-Micha Bodensohn, Ulf Brackmann, Liane Vogel, Matthias Urban, Anupam Sanghi, and Carsten Binnig. 2024. LLMs for Data Engineering on Enterprise Data. In *Proceedings of Workshops at the 50th International Conference on Very Large Data Bases, VLDB 2024, Guangzhou, China, August 26-30, 2024*. VLDB.org. https://vldb.org/workshops/2024/proceedings/TaDA/TaDA.4.pdf

[7] Avi Caciularu, Ido Dagan, Jacob Goldberger, and Arman Cohan. 2022. Long Context Question Answering via Supervised Contrastive Learning. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Seattle, United States, 2872–2879. doi:10.18653/v1/2022.naacl-main.207

[8] Zhiyu Cao and Zachary Feinstein. 2024. Large Language Model in Financial Regulatory Interpretation. arXiv:2405.06808 [q-fin.RM] https://arxiv.org/abs/2405.06808

[9] Alberto Caprara, Paolo Toth, and Matteo Fischetti. 2000. Algorithms for the Set Covering Problem. *Ann. Oper. Res.* 98, 1-4 (2000), 353–371. doi:10.1023/A%3A1019225027893

[10] Chi-Min Chan, Chunpu Xu, Ruibin Yuan, Hongyin Luo, Wei Xue, Yike Guo, and Jie Fu. 2024. Rq-rag: Learning to refine queries for retrieval augmented generation. *arXiv preprint arXiv:2404.00610* (2024).

[11] Edward Y. Chang and Longling Geng. 2025. SagaLLM: Context Management, Validation, and Transaction Guarantees for Multi-Agent LLM Planning. *Proc. VLDB Endow.* 18, 12 (Aug. 2025), 4874–4886. doi:10.14778/3750601.3750611

[12] Yangshen Deng and Zhengxin You. 2025. AlayaDB: The Data Foundation for Efficient and Effective Long-context LLM Inference. In *Companion of the 2025 International Conference on Management of Data* (Berlin, Germany) *(SIGMOD/PODS '25)*. Association for Computing Machinery, New York, NY, USA, 364–377. doi:10.1145/3722212.3724428

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Association for Computational Linguistics, 4171–4186. doi:10.18653/v1/N19-1423

[14] Darren Edge, Ha Trinh, Newman Cheng, Joshua Bradley, Alex Chao, Apurva Mody, Steven Truitt, Dasha Metropolitansky, Robert Osazuwa Ness, and Jonathan Larson. 2025. From Local to Global: A Graph RAG Approach to Query-Focused

Summarization. arXiv:2404.16130 [cs.CL] https://arxiv.org/abs/2404.16130

[15] Olivier Ferret. 1998. How to Thematically Segment Texts by using Lexical Cohesion?. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics, Volume 2*. Association for Computational Linguistics, Montreal, Quebec, Canada, 1481–1483. doi:10.3115/980691.980813

[16] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL] https://arxiv.org/abs/2312.10997

[17] Dorit S. Hochbaum. 1982. Approximation Algorithms for the Set Covering and Vertex Cover Problems. *SIAM J. Comput.* 11, 3 (1982), 555–556. doi:10.1137/0211045

[18] Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2023. Longllmlingua: Accelerating and enhancing llms in long context scenarios via prompt compression. *arXiv preprint arXiv:2310.06839* (2023).

[19] Ziyan Jiang, Xueguang Ma, and Wenhu Chen. 2024. LongRAG: Enhancing Retrieval-Augmented Generation with Long-context LLMs. arXiv:2406.15319 [cs.CL] https://arxiv.org/abs/2406.15319

[20] Chao Jin, Zili Zhang, Xuanlin Jiang, Fangyue Liu, Shufan Liu, Xuanzhe Liu, and Xin Jin. 2025. RAGCache: Efficient Knowledge Caching for Retrieval-Augmented Generation. 44, 1, Article 2 (Nov. 2025), 27 pages. doi:10.1145/3768628

[21] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 6769–6781. doi:10.18653/v1/2020.emnlp-main.550

[22] Arijit Khan, Tianxing Wu, and Xi Chen. 2024. LLM+KG: Data Management Opportunities in Unifying Large Language Models + Knowledge Graphs. In *Proceedings of Workshops at the 50th International Conference on Very Large Data Bases, VLDB 2024, Guangzhou, China, August 26-30, 2024*. VLDB.org. https://vldb.org/workshops/2024/proceedings/LLM+KG/LLM+KG-1.pdf

[23] Alex Kim, Maximilian Muhn, and Valeri Nikolaev. 2025. Financial Statement Analysis with Large Language Models. arXiv:2407.17866 [q-fin.ST] https://arxiv.org/abs/2407.17866

[24] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*. Association for Computing Machinery, New York, NY, USA, 611–626. doi:10.1145/3600006.3613165

[25] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

[26] LangChain. 2025. LangChain: The Platform for Reliable Agents. https://www.langchain.com/ Accessed: 2025-05-23.

[27] Quinn Leng, Jacob Portes, Sam Havens, Matei Zaharia, and Michael Carbin. 2024. Long Context RAG Performance of Large Language Models. *arXiv preprint arXiv:2411.03538* (2024). https://arxiv.org/abs/2411.03538

[28] 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. 2021. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. arXiv:2005.11401 [cs.CL] https://arxiv.org/abs/2005.11401

[29] 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. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *Advances in Neural Information Processing Systems*, Vol. 33. Curran Associates, Inc., 9459–9474.

[30] Guoliang Li, Xuanhe Zhou, and Xinyang Zhao. 2024. LLM for Data Management. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 4213–4216. doi:10.14778/3685800.3685838

[31] Qinbin Li, Junyuan Hong, Chulin Xie, Jeffrey Tan, Rachel Xin, Junyi Hou, Xavier Yin, Zhun Wang, Dan Hendrycks, Zhangyang Wang, Bo Li, Bingsheng He, and Dawn Song. 2024. LLM-PBE: Assessing Data Privacy in Large Language Models. *Proc. VLDB Endow.* 17, 11 (July 2024), 3201–3214. doi:10.14778/3681954.3681994

[32] Zhuoqun Li, Xuanang Chen, Haiyang Yu, Hongyu Lin, Yaojie Lu, Qiaoyu Tang, Fei Huang, Xianpei Han, Le Sun, and Yongbin Li. 2024. StructRAG: Boosting Knowledge Intensive Reasoning of LLMs via Inference-time Hybrid Information Structurization. arXiv:2410.08815 [cs.CL] https://arxiv.org/abs/2410.08815

[33] Zhuowan Li, Cheng Li, Mingyang Zhang, Qiaozhu Mei, and Michael Bendersky. 2024. Retrieval Augmented Generation or Long-Context LLMs? A Comprehensive Study and Hybrid Approach. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, Franck Dernoncourt, Daniel Preoţiuc-Pietro, and Anastasia Shimorina (Eds.). Association for Computational Linguistics, Miami, Florida, US, 881–893. doi:10.18653/v1/2024.emnlp-industry.66

[34] LlamaIndex. 2025. LlamaIndex: Build Knowledge Assistants over your Enterprise Data. https://www.llamaindex.ai Accessed: 2025-05-23.

[35] Songshuo Lu, Hua Wang, Yutian Rong, Zhi Chen, and Yaohua Tang. 2025. TurboRAG: Accelerating Retrieval-Augmented Generation with Precomputed KV Caches for Chunked Text. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Suzhou, China, 6588–6601. doi:10.18653/v1/2025.emnlp-main.334

[36] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query Rewriting for Retrieval-Augmented Large Language Models. arXiv:2305.14283 [cs.CL] https://arxiv.org/abs/2305.14283

[37] Meta AI. 2025. Llama: Inference Code for Llama Models. https://github.com/meta-llama/llama Accessed: 2025-05-23.

[38] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a Suit of Armor Conduct Electricity? A New Dataset for Open Book Question Answering. arXiv:1809.02789 [cs.CL] https://arxiv.org/abs/1809.02789

[39] Sewon Min, Victor Zhong, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2019. Multi-hop Reading Comprehension through Question Decomposition and Rescoring. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. Florence, Italy, 6097–6109. doi:10.18653/v1/P19-1613

[40] T. Mishra, E. Sutanto, R. Rossanti, et al. 2024. Use of large language models as artificial intelligence tools in academic research and publishing among global clinical researchers. *Scientific Reports* 14 (2024), 31672. doi:10.1038/s41598-024-81370-6

[41] Humza Naveed, Asad Ullah Khan, Shi Qiu, Muhammad Saqib, Saeed Anwar, Muhammad Usman, Naveed Akhtar, Nick Barnes, and Ajmal Mian. 2023. A Comprehensive Overview of Large Language Models. *arXiv preprint arXiv:2307.06435* (2023). https://arxiv.org/abs/2307.06435

[42] Rodrigo Nogueira and Kyunghyun Cho. 2020. Passage Re-ranking with BERT. arXiv:1901.04085 [cs.IR] https://arxiv.org/abs/1901.04085

[43] OpenAI. 2023. text-embedding-ada-002 Tokenizer. https://huggingface.co/Xenova/text-embedding-ada-002 Accessed: 2025-05-23.

[44] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[45] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 1532–1543. doi:10.3115/v1/D14-1162

[46] Danish Pruthi, Mansi Gupta, Bhuwan Dhingra, Graham Neubig, and Zachary C. Lipton. 2020. Learning to Deceive with Attention-Based Explanations. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, Online, 4782–4793. doi:10.18653/v1/2020.acl-main.432

[47] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. arXiv:2103.00020 [cs.CV] https://arxiv.org/abs/2103.00020

[48] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language Models are Unsupervised Multitask Learners. https://github.com/openai/gpt-2 Accessed: 2025-05-23.

[49] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. arXiv:1908.10084 [cs.CL] https://arxiv.org/abs/1908.10084

[50] Stephen Robertson. 2004. Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of documentation* 60, 5 (2004), 503–520.

[51] G. Salton, A. Wong, and C. S. Yang. 1975. A vector space model for automatic indexing. *Commun. ACM* 18, 11 (Nov. 1975), 613–620. doi:10.1145/361219.361220

[52] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. 2011. Finding a "Kneedle" in a Haystack: Detecting Knee Points in System Behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*. 166–171. doi:10.1109/ICDCSW.2011.20

[53] Dario Satriani, Enzo Veltri, Donatello Santoro, Sara Rosato, Simone Varriale, and Paolo Papotti. 2025. Logical and Physical Optimizations for SQL Query Execution over Large Language Models. *Proc. ACM Manag. Data* 3, 3, Article 181 (June 2025), 28 pages. doi:10.1145/3725411

[54] Tianyuan Shi, Liangzhi Li, Zijian Lin, Tao Yang, Xiaojun Quan, and Qifan Wang. 2023. Dual-feedback knowledge retrieval for task-oriented dialogue systems. *arXiv preprint arXiv:2310.14528* (2023).

[55] DailyDoseofDS Team. 2024. *5 Chunking Strategies For RAG*. Technical Report. Daily Dose of Data Science Blog. https://blog.dailydoseofds.com/p/5-chunking-strategies-for-rag

[56] G Team et al. 2023. Gemini: A Family of Highly Capable Multimodal Models. *arXiv preprint* arXiv:2312.11805 (2023). https://arxiv.org/abs/2312.11805

[57] V. A. Traag, L. Waltman, and N. J. van Eck. 2019. From Louvain to Leiden: guaranteeing well-connected communities. *Scientific Reports* 9, 1 (March 2019). doi:10.1038/s41598-019-41695-z

[58] Dietrich Trautmann, Alina Petrova, and Frank Schilder. 2022. Legal Prompt Engineering for Multilingual Legal Judgement Prediction. arXiv:2212.02199 [cs.CL] https://arxiv.org/abs/2212.02199

[59] Minzheng Wang, Longze Chen, Fu Cheng, Shengyi Liao, Xinghua Zhang, and et al. Wu. 2024. Leave No Document Behind: Benchmarking Long-Context LLMs with Extended Multi-Doc QA. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 5627–5646. doi:10.18653/v1/2024.emnlp-main.322

[60] Mengzhao Wang, Haotian Wu, Xiangyu Ke, Yunjun Gao, Xiaoliang Xu, and Lu Chen. 2024. An Interactive Multi-Modal Query Answering System with Retrieval-Augmented Large Language Models. *Proc. VLDB Endow.* 17, 12 (Aug. 2024), 4333–4336. doi:10.14778/3685800.3685868

[61] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems* (New Orleans, LA, USA). Curran Associates Inc., Red Hook, NY, USA, Article 1800, 14 pages.

[62] Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. Break It Down: A Question Understanding Benchmark. *Transactions of the Association for Computational Linguistics* 8 (2020), 183–198. doi:10.1162/tacl_a_00309

[63] Peng Xu, Wei Ping, Xianchao Wu, Lawrence McAfee, Chen Zhu, Zihan Liu, Sandeep Subramanian, Evelina Bakhturina, Mohammad Shoeybi, and Bryan Catanzaro. 2024. Retrieval meets Long Context Large Language Models. arXiv:2310.03025 [cs.CL] https://arxiv.org/abs/2310.03025

[64] An Yang, Baosong Yang, and Beichen Zhang. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] https://arxiv.org/abs/2412.15115

[65] Jiayi Yao, Hanchen Li, Yuhan Liu, Siddhant Ray, Yihua Cheng, Qizheng Zhang, Kuntai Du, Shan Lu, and Junchen Jiang. 2025. CacheBlend: Fast Large Language Model Serving for RAG with Cached Knowledge Fusion. In *Proceedings of the Twentieth European Conference on Computer Systems*. Association for Computing Machinery, New York, NY, USA, 94–109. doi:10.1145/3689031.3696098

[66] Jing Yao, Wei Xu, Jianxun Lian, Xiting Wang, Xiaoyuan Yi, and Xing Xie. 2023. Knowledge Plugins: Enhancing Large Language Models for Domain-Specific Recommendations. *arXiv preprint arXiv:2311.10779* (2023). https://arxiv.org/abs/2311.10779

[67] Fangyi Yu, Lee Quartey, and Frank Schilder. 2022. Legal Prompting: Teaching a Language Model to Think Like a Lawyer. arXiv:2212.01326 [cs.CL] https://arxiv.org/abs/2212.01326

[68] Haozhen Zhang, Tao Feng, and Jiaxuan You. 2025. Graph of Records: Boosting Retrieval Augmented Generation for Long-context Summarization with Graphs. ACL, Vienna, Austria, 23780–23799. doi:10.18653/v1/2025.acl-long.1159

[69] Haoyu Zhang, Jun Liu, Zhenhua Zhu, Shulin Zeng, Maojia Sheng, Tao Yang, Guohao Dai, and Yu Wang. 2024. Efficient and Effective Retrieval of Dense-Sparse Hybrid Vectors using Graph-based Approximate Nearest Neighbor Search. arXiv:2410.20381 [cs.IR] https://arxiv.org/abs/2410.20381

[70] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2025. A Survey of Large Language Models. arXiv:2303.18223 [cs.CL] https://arxiv.org/abs/2303.18223

[71] Xinyang Zhao, Xuanhe Zhou, and Guoliang Li. 2024. Chat2Data: An Interactive Data Analysis System with RAG, Vector Databases and LLMs. *Proc. VLDB Endow.* 17, 12 (2024), 4481–4484. doi:10.14778/3685800.3685905

[72] Huaixiu Steven Zheng, Swaroop Mishra, Xinyun Chen, Heng-Tze Cheng, Ed H. Chi, Quoc V Le, and Denny Zhou. 2024. Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models. arXiv:2310.06117 [cs.LG] https://arxiv.org/abs/2310.06117

[73] Yingli Zhou, Yaodong Su, Youran Sun, Shu Wang, Taotao Wang, Runyuan He, Yongwei Zhang, Sicong Liang, Xilin Liu, Yuchi Ma, and Yixiang Fang. 2025. In-depth Analysis of Graph-based RAG in a Unified Framework. arXiv:2503.04338 [cs.IR] https://arxiv.org/abs/2503.04338

[74] Shengyao Zhuang, Bing Liu, Bevan Koopman, and Guido Zuccon. 2023. Open-source large language models are strong zero-shot query likelihood models for document ranking. *arXiv preprint arXiv:2310.13243* (2023).

[75] Shengyao Zhuang, Bing Liu, Bevan Koopman, and Guido Zuccon. 2023. Open-source Large Language Models are Strong Zero-shot Query Likelihood Models for Document Ranking. Association for Computational Linguistics, Singapore, 8807–8817. doi:10.18653/v1/2023.findings-emnlp.590

Received October 2025; revised January 2026; accepted February 2026