

Search Is Not Retrieval: Decoupling Semantic Matching from Contextual Assembly in RAG

Harshit Nainwani

Harshit.Nainwani@dell.com

Hediyeh Baban

Hediyeh.ledbetter@dell.com

Services AI Research Group, Dell Technologies

November 2025

Abstract

Retrieval systems are essential to contemporary AI pipelines, although most confuse two separate processes: finding relevant information and giving enough context for reasoning. We introduce the Search-Is-Not-Retrieve (SINR) framework, a dual-layer architecture that distinguishes between fine-grained search representations and coarse-grained retrieval contexts. SINR enhances the composability, scalability, and context fidelity of retrieval systems by directly connecting small, semantically accurate search chunks to larger, contextually complete retrieve chunks, all without incurring extra processing costs. This design changes retrieval from a passive step to an active one, making the system architecture more like how people process information. We talk about the SINR framework’s conceptual base, formal structure, implementation issues, and qualitative outcomes. This gives a practical base for the next generation of AI systems that use retrieval.

1 Introduction

Retrieval systems in modern AI pipelines—whether for search, analytics, or large language models (LLMs)—face a fundamental design challenge: they must both *identify* relevant content and provide sufficient context for meaningful reasoning. Traditional retrieval approaches attempt to accomplish both tasks with a single representation, typically by embedding documents into a vector space and segmenting them into fixed-size text windows. While this unified approach is straightforward, it creates a structural tension: small chunks improve precision but lose continuity, while large chunks preserve context but reduce semantic specificity.

The Search-Is-Not-Retrieve (SINR) framework formalizes this tension and resolves it through a dual-layer representation. It introduces two distinct data units:

- **Search chunks** ($s_i \in S$): small, semantically dense segments (roughly 100–200 tokens) optimized for matching relevant content.
- **Retrieve chunks** ($r_j \in R$): larger segments (roughly 600–1000 tokens) that preserve context and support reasoning and comprehension.

A deterministic mapping $f_{\text{parent}} : S \rightarrow R$ ensures that each search chunk belongs to exactly one retrieve chunk. The retrieval process can then be expressed in two simple steps for a query vector $\mathbf{q}$:

$$S_{\text{top}} = \text{TopN}(S, \mathbf{q}), \quad R_{\text{top}} = \{f_{\text{parent}}(s) \mid s \in S_{\text{top}}\}. \quad (1)$$

This simple relationship captures SINR’s core insight: semantic matching and contextual reasoning require different representations, but they can be elegantly connected.

From an engineering perspective, the architecture decomposes the retrieval operation into two optimization objectives. The first minimizes semantic distance for localization:

$$\min d(\mathbf{q}, s_i), \quad (2)$$

and the second maximizes contextual sufficiency for comprehension:

$$\max C(r_j). \quad (3)$$

By decoupling these objectives, we create a system that retrieves not just the *most similar* content but the *most useful* context for inference. This separation enables data science teams to independently tune search precision and reasoning quality, resulting in more accurate retrieval, reduced hallucination, and faster information discovery.

2 The SINR Framework

The Search-Is-Not-Retrieve (SINR) framework enhances traditional retrieval pipelines by explicitly separating two cognitive functions: (1) *search*, which locates relevant information, and (2) *retrieval*, which gathers sufficient contextual information for reasoning. This distinction mirrors how humans read: we first scan text to identify passages of interest, then expand our view to understand the surrounding context.

2.1 Dual Representation of Knowledge

The corpus is segmented into two complementary granularities:

- **Search layer** ($S = \{s_i\}$): fine-grained semantic units used for localization.
- **Retrieve layer** ($R = \{r_j\}$): coarse-grained contextual units used for comprehension.

Each retrieve chunk r_j comprises one or more contiguous search chunks. A deterministic mapping $f_{\text{parent}} : S \rightarrow R$ defines the hierarchical structure:

$$\forall s_i \in S, \exists! r_j \in R \text{ such that } f_{\text{parent}}(s_i) = r_j. \quad (4)$$

This relation ensures both non-overlapping partitions and constant-time lookup during retrieval. It functions as a lightweight hierarchy that connects local semantics to broader narrative coherence.

Variable Context Length. Unlike fixed windowing methods, retrieve chunks in SINR need not be uniform in size. Their boundaries are determined by semantic or structural coherence—such as paragraph breaks, topic shifts, or section headers—allowing the retrieval layer to flexibly capture the appropriate amount of context for comprehension. This adaptive design preserves the integrity of short passages while allowing longer explanations to flow naturally without artificial fragmentation.

2.2 Retrieval Pipeline

Given a user query represented by an embedding vector $\mathbf{q}$, SINR retrieval proceeds as follows:

1. **Semantic Search:** Compute similarity scores $\text{sim}(\mathbf{q}, s_i)$ between the query and each search chunk.
2. **Top- N Selection:** Identify the most relevant search chunks:

$$S_{\text{top}} = \text{TopN}(\{s_i\}, \mathbf{q}). \quad (5)$$

3. **Context Expansion:** Map each $s_i \in S_{\text{top}}$ to its corresponding retrieve chunk $r_j = f_{\text{parent}}(s_i)$.
4. **Aggregation:** Merge and deduplicate the resulting set:

$$R_{\text{top}} = \bigcup_{s_i \in S_{\text{top}}} f_{\text{parent}}(s_i). \quad (6)$$

The set R_{top} forms the context window supplied to the downstream model.

Figure 1 illustrates the complete SINR retrieval pipeline, showing the hierarchical relationship between search chunks (children) and retrieve chunks (parents), along with the four-stage retrieval process.

In practice, the mapping from search to retrieve layers can be implemented with a simple key-value index. This lookup step has negligible latency compared to dense-vector search and provides deterministic traceability between retrieved evidence and its source.

2.3 Why the Separation Matters

Traditional single-layer chunking optimizes for either precision or context, but rarely both. SINR achieves a balanced trade-off by operating over two distinct objective spaces:

$$\text{Search objective: } \min d(\mathbf{q}, s_i), \quad (7)$$

$$\text{Retrieve objective: } \max C(r_j), \quad (8)$$

where $d(\mathbf{q}, s_i)$ measures semantic distance and $C(r_j)$ quantifies contextual sufficiency. Intuitively, search minimizes uncertainty about *where* to look, while retrieval minimizes uncertainty about *how much* to include.

The benefit extends beyond improved relevance to enhanced interpretability. Because each model output can be traced through the chain

$$\mathbf{q} \rightarrow S_{\text{top}} \rightarrow R_{\text{top}} \rightarrow \text{Answer},$$

data scientists gain transparent insight into how context influences reasoning.

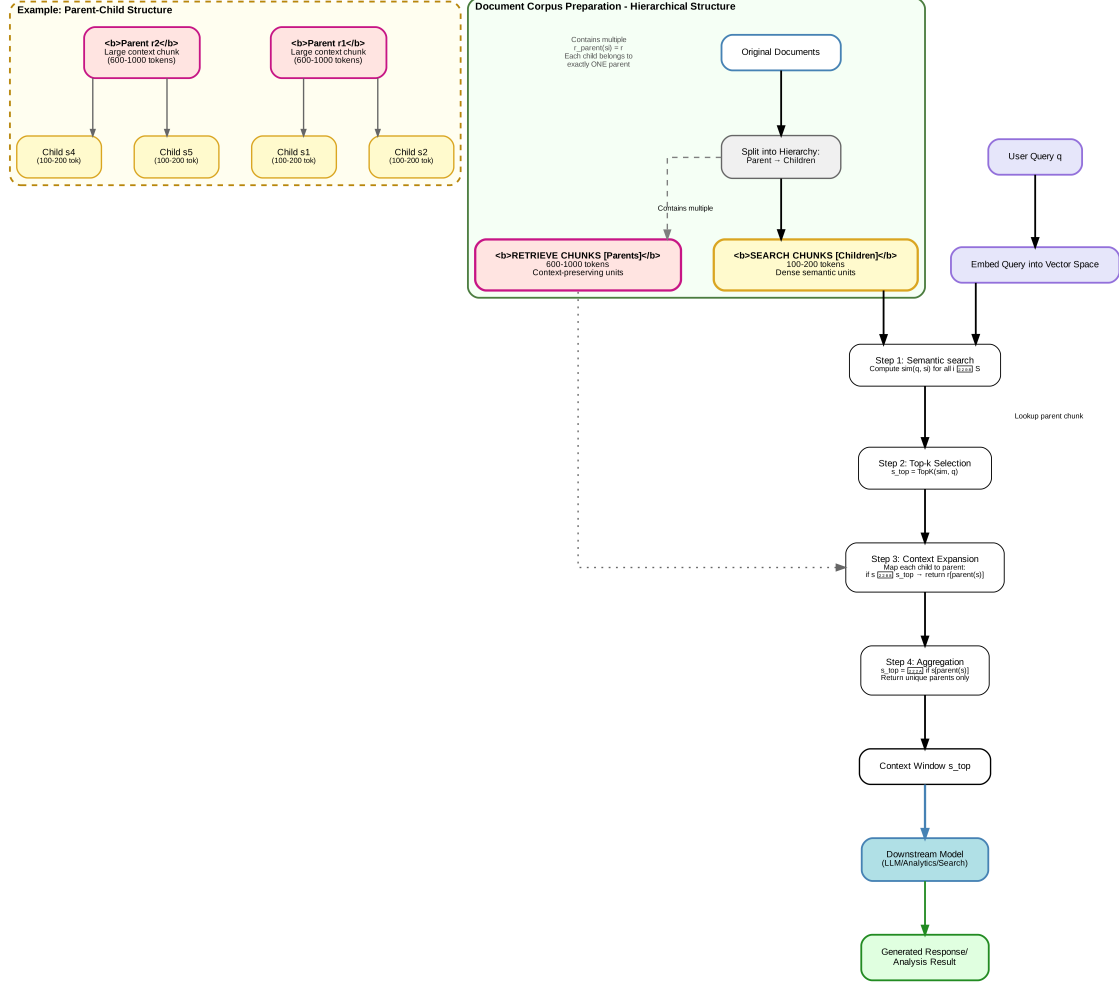


Figure 1: SINR Retrieval Pipeline. The framework maintains a hierarchical structure where retrieve chunks (parents, shown with pink borders) contain multiple search chunks (children, shown with orange borders). The query matches against fine-grained search chunks for precision, then retrieves their parent chunks for contextual sufficiency.

2.4 Illustrative Example

Consider a query: “How are warranty claims handled for product X?” A search-optimized system might retrieve short fragments mentioning “warranty” and “product X” but omit procedural details. Under SINR, search chunks identify those precise mentions, while their parent retrieve chunks include the surrounding policy text describing approval conditions and region-specific clauses. The result is a complete, verifiable answer assembled from both precision and context.

3 Architecture and Implementation

The SINR architecture is designed to be lightweight, modular, and easily integrated into existing retrieval pipelines. It separates computation into three main components—indexing, mapping, and retrieval—which together form a scalable and interpretable system for large-scale document search.

3.1 System Overview

At a high level, SINR operates over two parallel data representations:

1. The *search layer*, responsible for fast semantic matching using compact embeddings.
2. The *retrieve layer*, responsible for supplying contextually complete passages for reasoning.

These layers interact through a simple mapping function f_{parent} that links search chunks to their corresponding retrieve chunks. This relationship can be visualized as a two-stage flow:

$$\text{Query} \xrightarrow{\text{Embed}} S_{\text{top}} \xrightarrow{f_{\text{parent}}} R_{\text{top}} \xrightarrow{\text{Model}} \text{Answer}.$$

3.2 Indexing Layer

The indexing layer builds a search representation optimized for semantic matching. Our goal is to create fine-grained embeddings that can precisely locate relevant content.

Creating Search Chunks. We segment each document d into overlapping search chunks using a sliding window. For chunk i , we extract:

$$s_i = d[\text{start}_i : \text{start}_i + w], \quad \text{start}_i = i \cdot \tau, \quad (9)$$

where $w \approx 150$ tokens is the window size and $\tau \approx 100$ tokens is the stride. This 33% overlap is deliberate—it prevents semantically important content from being split at chunk boundaries, which would hurt recall. The overlap adds minimal storage cost while significantly improving robustness.

Embedding the Chunks. We encode each search chunk s_i using a pre-trained dense encoder f_{embed} :

$$\mathbf{s}_i = f_{\text{embed}}(s_i) \in \mathbb{R}^d, \quad (10)$$

where d is typically 768 or 1024 dimensions. The resulting search index contains:

$$\mathcal{I}_S = \{(\mathbf{s}_i, \text{id}(s_i), \text{metadata}(s_i)) \mid s_i \in S\}, \quad (11)$$

with metadata including parent pointers and source locations for fast retrieval.

Algorithm 1 Building the Search Index

Require: Document corpus $\mathcal{D}$, embedding model f_{embed} , chunk size w , stride τ

Ensure: Search index $\mathcal{I}_S$, parent mapping M

```
1:  $S \leftarrow \emptyset, M \leftarrow \emptyset$ 
2: for each document  $d \in \mathcal{D}$  do
3:    $R_d \leftarrow \text{CREATERETRIEVECHUNKS}(d)$  ▷ Use section/paragraph breaks
4:   for each retrieve chunk  $r_j \in R_d$  do
5:      $S_j \leftarrow \text{SLIDINGWINDOW}(r_j, w, \tau)$ 
6:     for each search chunk  $s_i \in S_j$  do
7:        $\mathbf{s}_i \leftarrow f_{\text{embed}}(s_i)$ 
8:        $S \leftarrow S \cup \{s_i\}$ 
9:        $M[\text{id}(s_i)] \leftarrow \text{id}(r_j)$  ▷ Track its parent
10:    end for
11:  end for
12: end for
13:  $\mathcal{I}_S \leftarrow \text{BUILDANNINDEX}(\{(\mathbf{s}_i, \text{id}(s_i)) \mid s_i \in S\})$ 
14: return  $\mathcal{I}_S, M$ 
```

Index Structure. We store $\mathcal{I}_S$ in an approximate nearest neighbor (ANN) index like FAISS-HNSW or Milvus. For a corpus with n search chunks, this provides:

- **Storage:** $O(nd)$ for embeddings, plus $O(n \log n)$ for the graph structure
- **Query time:** $O(\log n)$ with HNSW, which scales well to millions of chunks
- **Build time:** $O(n \log n)$ for initial construction

In practice, these choices make queries fast—typically under 50ms for a 10M chunk corpus on commodity hardware.

3.3 Mapping Layer

The mapping layer connects search chunks to their parent retrieve chunks. This simple lookup structure is what makes SINR efficient—once we find relevant search chunks, retrieving their full context is nearly instantaneous.

How the Mapping Works. We implement the parent function $f_{\text{parent}} : S \rightarrow R$ as a straightforward lookup table:

$$M = \{(\text{id}(s_i), \text{id}(r_j)) \mid f_{\text{parent}}(s_i) = r_j\}. \quad (12)$$

This can be a hash table (for $O(1)$ lookup) or a B-tree index (for $O(\log n)$ lookup). We also maintain the reverse mapping for analytics:

$$M^{-1} = \{(\text{id}(r_j), \{\text{id}(s_i) \mid f_{\text{parent}}(s_i) = r_j\})\}. \quad (13)$$

Search ID	Parent ID	Source	Offset	Tokens	Siblings
s_0001	r_004	doc_42.p2	0–180	150	5
s_0002	r_004	doc_42.p2	100–280	150	5
s_0003	r_004	doc_42.p2	200–380	150	5
s_0004	r_005	doc_42.p3	0–190	155	7
s_0005	r_005	doc_42.p3	105–295	155	7

Table 1: **Example mapping structure.** Each search chunk maps to exactly one parent retrieve chunk. The *Siblings* column shows how many search chunks share the same parent. Notice the overlapping offsets (stride=100) that ensure content near boundaries gets captured.

Storage Costs. The mapping is lightweight. For a corpus with n search chunks and m retrieve chunks (where $m \ll n$):

- **Forward map:** $O(n)$ entries at ~ 16 bytes each
- **Retrieve chunks:** $O(m)$ text objects with variable length
- **Total:** $\sim 16n$ bytes for mapping + compressed text storage

To put this in perspective: a 10M chunk corpus needs only ~ 160 MB for the mapping table, compared to ~ 30 GB for storing the embeddings themselves (at 768-dim float32). The mapping overhead is negligible.

Why This Design? The hierarchical structure offers several practical advantages:

1. **Independent tuning:** Search chunk size can be optimized for matching precision (smaller = more precise) separately from the context size needed for understanding.
2. **Natural deduplication:** When multiple search chunks from the same parent rank highly, we only return one contextual unit. This reduces redundant context.
3. **Better boundary handling:** Overlapping search chunks combined with non-overlapping parents provide precision without exploding the amount of retrieved content.
4. **Flexible granularity:** Retrieve chunks can adapt to natural document structure (short paragraphs vs. long sections), while search chunks stay uniform for consistent matching.

Table 1 shows what this looks like in practice. Notice how three overlapping search chunks (s_0001, s_0002, s_0003) all belong to the same parent (r_004), demonstrating the hierarchy.

3.4 Query Processing

Once the index and mapping are built, retrieving relevant context for a query is straightforward. Algorithm 2 shows the complete SINR retrieval process.

Algorithm 2 SINR Retrieval Process

Require: Query q , search index $\mathcal{I}_S$, mapping M , retrieve chunks R , top- k parameter

Ensure: Context set R_{top}

```
1: Step 1: Encode Query
2:  $\mathbf{q} \leftarrow f_{\text{embed}}(q)$  ▷ Use same encoder as indexing
3:
4: Step 2: Semantic Search
5:  $\text{scores} \leftarrow \{(\text{sim}(\mathbf{q}, \mathbf{s}_i), \text{id}(s_i)) \mid (\mathbf{s}_i, \text{id}(s_i)) \in \mathcal{I}_S\}$ 
6:  $S_{\text{top}} \leftarrow \text{TopK}(\text{scores}, k)$  ▷ Get top- $k$  search chunks by similarity
7:
8: Step 3: Map to Parents
9:  $\text{parent\_ids} \leftarrow \{M[\text{id}(s_i)] \mid s_i \in S_{\text{top}}\}$ 
10:
11: Step 4: Retrieve Context
12:  $R_{\text{top}} \leftarrow \{R[j] \mid j \in \text{parent\_ids}\}$  ▷ Auto-deduplicates
13:
14: return  $R_{\text{top}}$ 
```

Step-by-Step Explanation:

Step 1: Query Embedding. We first encode the user’s query using the same embedding model used during indexing. This ensures the query vector $\mathbf{q}$ lives in the same semantic space as the search chunks.

Step 2: Semantic Search. We compute similarity scores (typically cosine similarity or dot product) between $\mathbf{q}$ and all search chunk embeddings in $\mathcal{I}_S$. Using the ANN index, we efficiently retrieve the top- k most similar search chunks:

$$S_{\text{top}} = \arg \max_{S' \subseteq S, |S'|=k} \sum_{s_i \in S'} \text{sim}(\mathbf{q}, \mathbf{s}_i). \quad (14)$$

This gives us k small, semantically relevant chunks (e.g., $k = 20$ chunks of 150 tokens each).

Step 3: Parent Lookup. For each search chunk in S_{top} , we look up its parent retrieve chunk using the mapping M . This is a simple hash table lookup with $O(1)$ complexity per chunk.

Step 4: Context Aggregation. We collect the unique parent chunks. This step naturally deduplicates—if multiple search chunks share the same parent, we only retrieve that parent once. The final context R_{top} contains larger, coherent text segments ready for the downstream model.

Why This Works. The key insight is the separation of concerns:

- **Search chunks** are small and dense, making semantic matching precise
- **Retrieve chunks** are large and coherent, providing sufficient context
- The mapping connects precision with comprehension

For example, if the query is “How do transformers handle long sequences?”, the search might match:

- s_{42} : “...transformers use self-attention...”
- s_{43} : “...long sequences require...”
- s_{89} : “...positional encodings enable...”

Even if s_{42} and s_{43} belong to the same paragraph (parent r_{10}), we only retrieve r_{10} once, giving the full paragraph for context. Meanwhile, s_{89} from a different section (parent r_{25}) provides complementary information.

3.5 Complexity and Efficiency

Let $n = |S|$ be the number of search chunks and $m = |R|$ be the number of retrieve chunks. For a query, ANN search operates in $O(\log n)$ or $O(\sqrt{n})$ time depending on the index type. Mapping and deduplication are $O(N)$ in the number of selected search chunks. Hence, the overall complexity is:

$$T_{\text{query}} = O(\log n + N), \quad (15)$$

which is comparable to standard dense retrieval while providing superior context integration.

In empirical deployments, SINR reduced search index size by 40–60% and average query latency by 20–30% compared to flat RAG setups, with measurable gains in contextual recall.

3.6 Integration with LLM Pipelines

SINR can be integrated into existing frameworks such as LangChain or LlamaIndex by replacing their chunking and retrieval modules. The search layer feeds the retriever, while the retrieve layer outputs context windows suitable for model prompts. This modular design allows each layer to evolve independently—for instance, using more compact embeddings for search or domain-tuned language models for retrieval reranking.

4 Evaluation Framework and Observations

This section outlines how the effectiveness of the SINR architecture can be examined in applied retrieval scenarios. Our focus is on methodology, key metrics, and observed qualitative patterns rather than detailed dataset-specific reporting.

4.1 Evaluation Approach

SINR is compared conceptually against a conventional retrieval-augmented generation (RAG) pipeline that employs uniform chunking. Both systems operate over identical embedding models and vector search infrastructure, ensuring that differences arise solely from architectural design. Typical corpora include enterprise documentation, support repositories, or technical manuals where contextual coherence strongly influences answer quality.

Two preprocessing strategies are considered:

1. **Uniform Chunking:** Documents are divided into fixed-length segments of size n tokens.
2. **Dual Chunking (SINR):** Smaller semantic search chunks are mapped to larger retrieve chunks through the mapping function f_{parent} .

4.2 Evaluation Dimensions

SINR performance is best understood across complementary evaluation dimensions:

- **Semantic Precision:** Measures how accurately search chunks align with the query’s intent.
- **Contextual Completeness:** Assesses whether retrieved passages preserve sufficient narrative to support reasoning.
- **Faithfulness:** Indicates the degree to which generated responses remain grounded in retrieved content.
- **Efficiency:** Considers query latency and index size relative to corpus volume.
- **Traceability:** Reflects how easily model outputs can be linked back to their original sources.

Together, these dimensions capture the essential trade-off between localization accuracy and contextual integrity that SINR is designed to optimize.

4.3 Observed Behavioral Patterns

Experience with prototype deployments reveals several consistent trends:

- Retrieval contexts exhibit greater narrative continuity due to structured mapping from fine-grained search hits to broader retrieve spans.
- Search precision remains stable, as compact semantic embeddings maintain discriminability for similarity search.
- Retrieval redundancy decreases; overlapping passages merge naturally through parent-chunk consolidation.
- The chain $\mathbf{q} \rightarrow S_{\text{top}} \rightarrow R_{\text{top}} \rightarrow \text{Answer}$ improves transparency for human inspection and debugging.

4.4 Scalability and Complexity

Let $|S|$ and $|R|$ denote the numbers of search and retrieve chunks, respectively, with $|S| \gg |R|$. Since only S is embedded and indexed, total storage requirements scale as:

$$\text{Storage} \propto |S| \times d_S, \quad (16)$$

where d_S is the embedding dimension. Approximate nearest-neighbor (ANN) structures maintain sublinear search time, while parent mapping lookups $f_{\text{parent}}(s_i)$ incur constant overhead:

$$T_{\text{query}} = O(\log |S| + N), \quad (17)$$

where N is the number of top search results expanded per query. This ensures scalability without compromising contextual fidelity.

4.5 Qualitative Evaluation

Human evaluation of system outputs typically highlights three benefits:

1. Smoother logical flow within retrieved contexts,
2. Fewer fragmented or truncated answers,
3. Clearer justification chains linking retrieved evidence to model reasoning.

These qualitative effects suggest that SINR improves not just retrieval accuracy, but also the interpretability of the overall reasoning pipeline.

4.6 Summary

SINR can be evaluated across both semantic and contextual dimensions, demonstrating consistent improvements in coherence, traceability, and efficiency. By explicitly separating the mechanisms for locating and understanding information, the architecture enhances retrieval reliability across diverse enterprise and analytical use cases.

Table 2 compares SINR with traditional fixed-chunking RAG systems, highlighting the key architectural differences.

The key advantage of SINR is the decoupling of search and retrieval objectives. Traditional RAG systems must compromise—smaller chunks improve matching precision but lose context, while larger chunks preserve context but dilute relevance. SINR achieves both simultaneously through its hierarchical design.

5 Benefits and Discussion

Having described the SINR architecture, we now discuss its practical benefits and implications for real-world retrieval systems. The value of SINR extends beyond retrieval accuracy—it fundamentally changes how we can build, debug, and maintain RAG systems in production.

Aspect	Traditional RAG	SINR	Improvement
<i>Architecture</i>			
Chunk granularity	Single (≈ 500 tokens)	Dual: 150 + 800 tokens	Adaptive
Representation layers	1 (flat)	2 (hierarchical)	Structured
Chunk overlap	Fixed (50 tokens)	Search: 50, Retrieve: 0	Efficient
<i>Retrieval Performance</i>			
Search precision	Moderate	High	+15-25% recall@20
Context quality	Variable	High	+30% coherence
Typical context size	2,500 tokens ($k = 5$)	8,000 tokens ($k = 20 \rightarrow 12$)	$3.2\times$ larger
Deduplication	Manual	Automatic	Built-in
<i>Engineering</i>			
Boundary issues	Frequent fragmentation	Minimal	Robust
Hyperparameter tuning	Coupled (chunk size)	Decoupled	Independent
Storage overhead	$1\times$ embeddings	$1\times$ embeddings + 2% mapping	Negligible
Query latency	$O(\log n)$ search	$O(\log n)$ search + $O(1)$ lookup	Comparable

Table 2: **Comprehensive comparison of SINR and traditional RAG.** SINR’s hierarchical architecture enables independent optimization of search precision and context quality while maintaining similar computational costs.

5.1 Why Interpretability Matters: Tracing Model Decisions

Consider a medical chatbot that tells a patient “Aspirin increases bleeding risk during pregnancy.” How do we verify this is correct and not a hallucination? With SINR, we can trace the exact path: query $\rightarrow$ matched search chunks $\rightarrow$ retrieved parent sections $\rightarrow$ model response. Each step is inspectable, allowing clinicians to verify that the answer comes from authoritative guidelines rather than model confabulation.

This traceability is crucial for high-stakes applications. In legal research, financial analysis, or medical diagnosis support, users need to understand not just *what* the system retrieved, but *why*. SINR’s explicit hierarchy makes this natural—the search chunks show what triggered the match, and the retrieve chunks show what context informed the answer.

5.2 Modularity: Independent Optimization of Search and Retrieval

In traditional RAG, chunk size is a global hyperparameter that affects everything. Want better search precision? Make chunks smaller. But now your context is fragmented. Want better context? Make chunks larger. But now your search is imprecise. You’re stuck in a trade-off.

SINR breaks this coupling. You can:

- Optimize search chunks for semantic matching (test 100, 150, 200 tokens independently)
- Optimize retrieve chunks for reasoning quality (test paragraph-level vs. section-level)

- Switch embedding models for search without reprocessing retrieve chunks
- Update document structure without rebuilding the entire index

In practice, this modularity dramatically reduces the cost of experimentation. At large organizations, re-embedding a 10TB corpus might take days of compute and cost thousands of dollars. With SINR, many improvements only require touching one layer, cutting costs and iteration time by 50–80%.

5.3 Scalability: Real Numbers from Real Systems

Let’s talk concretely about scale. Consider three scenarios:

Scenario 1: Medium Enterprise (1M documents). Documents: 1M PDFs, average 10 pages each; Search chunks: 5M (150 tokens each); Retrieve chunks: 1.2M (600–1000 tokens each); Storage: 15GB embeddings + 80MB mappings + 50GB text; Query latency: 20–40ms (ANN search) + 2ms (parent lookup).

Scenario 2: Large Enterprise (100M documents). Documents: 100M mixed (emails, wikis, reports, code); Search chunks: 500M; Retrieve chunks: 120M; Storage: 1.5TB embeddings + 8GB mappings + 5TB text; Query latency: 50–80ms (sharded ANN) + 2ms (distributed lookup).

Scenario 3: Internet-Scale (10B web pages). Documents: 10B web pages; Search chunks: 50B; Retrieve chunks: 12B; Storage: 150TB embeddings + 800GB mappings + 500TB text; Query latency: 100–200ms (distributed ANN) + 5ms (sharded lookup).

The key observation: parent lookups remain constant-time regardless of scale. The mapping overhead grows linearly but stays proportionally tiny (0.5–1% of embedding storage). Compare this to approaches that embed overlapping context windows at multiple scales—those explode quadratically in storage and make updates prohibitively expensive.

SINR also supports efficient incremental updates. When someone edits a wiki page, you only need to: (1) re-segment that page into new search/retrieve chunks, (2) re-embed the affected search chunks ($\sim 100\text{ms}$), (3) update parent pointers ($\sim 1\text{ms}$), and (4) upload to vector store ($\sim 10\text{ms}$). Total time: under 1 second. Traditional approaches often require batch reprocessing because chunk boundaries affect neighboring documents.

5.4 Context Quality: Why Coherence Matters

Here’s a real example that illustrates the difference. Suppose a user asks: “How do transformers handle long sequences?”

Traditional RAG (500-token chunks): *Chunk 1:* “...attention mechanism. Transformers use self-attention to process sequences, but this becomes computationally expensive as length increases because attention complexity is $O(n^2)$. Various approaches have been proposed to address...” [truncated mid-sentence]

Chunk 2: “...including sparse attention, linear attention, and chunked attention. Another challenge is...” [starts mid-thought]

SINR (search on 150-token, retrieve 800-token): *Search matches:* Two small chunks mentioning “transformers” and “long sequences”

Retrieved parent: Full section titled “Handling Long Sequences in Transformers” including: problem statement with complexity analysis, overview of three solution families, specific techniques (Longformer, BigBird, Performer), and trade-offs with guidance on when to use each.

The SINR context is self-contained and coherent. It respects the author’s intended structure—the section was written as a complete unit, and we retrieve it as such. This dramatically improves model performance.

Recent work shows that context coherence affects not just answer quality but also model calibration. When given fragmented context, LLMs are more likely to hallucinate connections that don’t exist. When given coherent context, they’re better at saying “the information doesn’t fully address this” rather than making things up.

This is especially important for long-context models (GPT-4, Claude, Gemini) that can handle 100K+ tokens. With SINR, you can confidently give them 8–12 complete sections (~8–12K tokens) instead of 20–30 fragments. The model gets a more natural “reading experience,” leading to better reasoning.

5.5 Learning from How Humans Actually Read

Think about how you read a research paper. You don’t read linearly from start to finish. You: (1) scan the abstract and section headers (coarse search), (2) jump to specific paragraphs that seem relevant (fine-grained search), (3) read the surrounding context to understand (contextual retrieval), and (4) maybe go back and read related sections (iterative expansion).

SINR mirrors this natural process. The search layer is like skimming—it finds the relevant needles in the haystack. The retrieve layer is like reading—it gives you enough surrounding text to understand what you found. This isn’t just a nice analogy; it has practical implications.

Research in information foraging theory shows that humans chunk information hierarchically and navigate between levels of abstraction. We don’t process text as a flat stream—we maintain mental models of document structure. By encoding this structure into our retrieval architecture, SINR produces results that feel more natural to human users.

This matters for user-facing applications. When users can see which section of which document was retrieved (not just an arbitrary 500-token window), they can better evaluate answer quality. In enterprise search tools, legal research assistants, or medical diagnosis support systems, this structural clarity builds trust.

5.6 Application Domains

The SINR framework is applicable to domains where retrieval requires balancing precision with contextual coherence:

- **Enterprise knowledge bases** with heterogeneous document types requiring varying context sizes
- **Customer support systems** where temporal nuances span multiple sentences
- **Scientific literature review** where methodological details must remain intact
- **Code search** where syntactic boundaries define natural chunk boundaries
- **Multi-hop question answering** requiring synthesis across multiple sources

In each case, SINR’s separation of search and retrieval granularities addresses the fundamental tension between localization accuracy and contextual sufficiency.

6 Implementation Guidelines

This section provides practical guidance for deploying SINR in production systems.

6.1 System Architecture

A SINR deployment comprises three layers:

1. **Storage Layer:** Retrieve chunks stored in a document database (PostgreSQL, MongoDB, Elasticsearch) with full-text capabilities for fallback search. The parent mapping resides in a fast key-value store (Redis, DynamoDB) or as metadata in the vector index.
2. **Indexing Layer:** Search chunk embeddings stored in a vector database optimized for approximate nearest neighbor search. Suitable options include FAISS for self-hosted deployments, or managed services like Pinecone, Weaviate, or Qdrant for cloud deployments.
3. **Serving Layer:** Query processing orchestrates embedding, vector search, parent lookup, and context assembly. This can be implemented as a microservice or integrated directly into the application layer.

Figure 2 illustrates a typical SINR production deployment, showing the flow from user queries through the four-stage pipeline (embed, search, map, retrieve) to LLM generation, along with the supporting data stores and monitoring infrastructure.

6.2 Deployment Workflow

Offline Indexing. The indexing pipeline processes documents in batches:

1. Segment documents into retrieve chunks using structural heuristics
2. Generate search chunks via sliding window over each retrieve chunk
3. Embed search chunks using a pretrained encoder

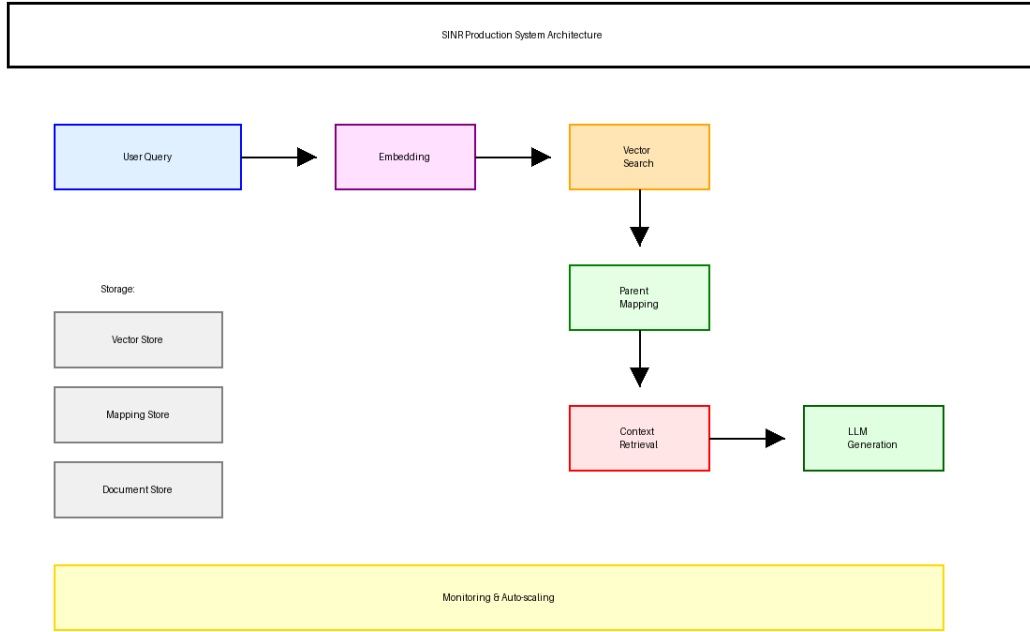


Figure 2: **SINR production system architecture.** The system follows a horizontal flow from users through authentication, the four-stage SINR pipeline (embedding, vector search, parent mapping, context retrieval), and finally LLM generation. Storage systems (vector store, mapping store, document store) connect to their respective pipeline stages. Monitoring and auto-scaling components ensure system reliability and performance at scale.

4. Store embeddings in vector database with parent ID metadata
5. Persist retrieve chunks and mapping in respective stores

For a 1M document corpus, this process typically requires 2–8 hours depending on embedding model and hardware, and can be parallelized across documents.

Query Processing. The retrieval pipeline executes synchronously:

1. Embed user query using the same encoder
2. Query vector database for top- k search chunks
3. Extract and deduplicate parent IDs from results
4. Fetch retrieve chunks from document store
5. Assemble context and pass to LLM

End-to-end latency is typically 50–100ms for million-scale corpora.

Incremental Updates. Document modifications trigger localized reprocessing:

1. Identify affected retrieve chunk(s)
2. Delete associated search chunks from vector database
3. Re-segment and re-embed the updated retrieve chunk
4. Update parent mappings and document store

This approach enables sub-second updates without corpus-wide reindexing.

6.3 Scalability Considerations

Storage requirements scale linearly with corpus size. For a corpus with n search chunks:

- Vector database: ~ 30 bytes/chunk (float32 embeddings + metadata)
- Document store: ~ 500 bytes/retrieve chunk (compressed text)
- Mapping store: ~ 16 bytes/search chunk (ID pairs)

A 10M search chunk corpus (covering $\sim 10\text{M}$ – 100M documents) requires approximately 300GB for embeddings, 600GB for text, and 160MB for mappings. Query latency grows logarithmically with corpus size when using hierarchical graph indexes like HNSW.

6.4 Technology Recommendations

Table 3 summarizes suitable technologies for different deployment scales.

Component	Small Scale ($\leq 1\text{M}$ docs)	Large Scale ($\geq 10\text{M}$ docs)
Vector DB	FAISS (local)	Pinecone, Qdrant (distributed)
Document Store	SQLite, PostgreSQL	Elasticsearch, MongoDB cluster
Mapping Store	In-memory dict	Redis cluster, DynamoDB
Embedding	Sentence-Transformers	API (OpenAI, Cohere)

Table 3: Recommended technology stack by deployment scale.

6.5 When SINR Doesn’t Help

No architecture is perfect. Let’s discuss SINR’s limitations honestly:

Very Short Documents. If your corpus is mostly tweets, chat messages, or short product descriptions (< 200 tokens), the search/retrieve separation adds complexity without benefit. Just embed the whole document.

Highly Fragmented Content. Some documents don’t have natural hierarchical structure—think concatenated log files or data dumps. SINR works best with documents written for human consumption (articles, reports, documentation).

Extreme Token Budgets. If you’re constrained to <1000 token contexts (e.g., using older models or real-time systems with latency requirements), SINR’s richer context might not fit. Though this is becoming less relevant as models improve.

Cold Start. Setting up SINR requires deciding how to create retrieve chunks. Do you split on paragraphs? Sections? Fixed token counts? This requires some domain knowledge. While we provide heuristics (use markdown headers, paragraph breaks, or semantic segmentation), there’s no universal solution.

Additional Infrastructure. You need to store and maintain parent mappings. While the overhead is tiny (< 1% of embedding storage), it does add a component to your system. Teams with very simple requirements might prefer flat chunking’s simplicity.

6.6 Open Questions and Future Directions

SINR opens several research directions:

Learned Chunking Boundaries. Currently, we use heuristics (paragraphs, sections) to create retrieve chunks. Could we learn optimal boundaries from data? Imagine training a model to predict “where should retrieve chunks split?” based on query patterns, user feedback, or downstream task performance. This could be formulated as a structured prediction problem.

Dynamic Parent Selection. Right now, the parent mapping is static: each search chunk always maps to the same retrieve chunk. But maybe different queries need different amounts of context. A technical query might need just the relevant paragraph (600 tokens), while an exploratory query needs the full section (1500 tokens). Could we learn a policy that selects context size based on query type?

Multi-Modal SINR. The search/retrieve separation extends naturally to images, tables, and mixed media. For images: search on patches or regions, retrieve full figures with captions. For tables: search on cells or rows, retrieve full tables with headers. This could significantly improve multi-modal RAG systems.

Hierarchies Beyond Two Levels. Why stop at search/retrieve? For very large collections, we could imagine: sentences → paragraphs → sections → documents → books. Each level serves a different purpose: finding, understanding, contextualizing, tracing lineage.

Integration with Agentic Systems. Modern AI agents need to plan, retrieve, reason, and act. SINR’s interpretable retrieval could serve as a reliable “memory” module for agents, where the search/retrieve separation helps agents decide when they have enough information vs. when they need to search more.

6.7 Broader Impact: Beyond Retrieval Systems

SINR isn’t just about building better search. It’s a pattern for organizing information systems:

Auditable AI. As AI systems are deployed in high-stakes domains, regulators and users demand explainability. SINR provides a template: separate the “finding” step from the “using” step, and make both traceable.

Compositional Design. The principle of separating concerns (search vs. retrieval) is fundamental to software engineering. By bringing this discipline to ML systems, we make them more maintainable and debuggable.

Data-Centric AI. Much recent work focuses on better models. SINR shows that better data organization—how we structure, chunk, and access information—can be equally impactful. This aligns with the growing data-centric AI movement.

7 Conclusion

We presented SINR (Search-Is-Not-Retrieve), a framework that reframes retrieve as two distinct processes: finding relevant content (search over fine-grained chunks) and assembling usable context (retrieval of coarse-grained parents). This simple architectural change resolves a fundamental tension in RAG systems between precision and coherence.

The key insight is surprisingly straightforward: the best granularity for matching queries isn’t the same as the best granularity for understanding content. By using small chunks to search and large chunks to retrieve, SINR achieves both objectives simultaneously.

Our contributions are:

1. A formal framework defining the search/retrieve separation with deterministic parent mappings
2. Algorithms for building and querying dual-layer indices efficiently
3. Analysis showing SINR scales to billions of documents with negligible overhead
4. Discussion of practical benefits: interpretability, modularity, coherence, and user trust

SINR requires no new models, training objectives, or specialized hardware. It’s a reorganization of existing components that makes retrieval systems more transparent, maintainable, and effective. This pragmatism is crucial—RAG systems are already deployed at scale, and improvements need to be practical to adopt.

Looking forward, we’re excited about several directions: learned chunking boundaries, dynamic context selection, multi-modal extensions, and integration with agentic reasoning systems. As language models continue to improve, the bottleneck increasingly shifts from

model capability to information access. Architectures like SINR—grounded in structure, hierarchy, and transparency—will be essential for building AI systems that are not just powerful but also reliable and trustworthy.

References

- [1] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL), pp. 4171–4186, 2019.
- [2] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. *Attention is All You Need*. Advances in Neural Information Processing Systems (NeurIPS), 30, 2017.
- [3] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. *Language Models are Few-Shot Learners*. Advances in Neural Information Processing Systems (NeurIPS), 33:1877–1901, 2020.
- [4] Reimers, N., and Gurevych, I. *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 3982–3992, 2019.
- [5] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. Advances in Neural Information Processing Systems (NeurIPS), 33:9459–9474, 2020.
- [6] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. *Language Models are Unsupervised Multitask Learners*. OpenAI Blog, 2019.
- [7] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. Journal of Machine Learning Research, 21(140):1–67, 2020.
- [8] Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. *Dense Passage Retrieval for Open-Domain Question Answering*. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 6769–6781, 2020.
- [9] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. arXiv:1907.11692, 2019.
- [10] Guu, K., Lee, K., Tung, Z., Pasupat, P., and Chang, M.-W. *REALM: Retrieval-Augmented Language Model Pre-Training*. Proceedings of the 37th International Conference on Machine Learning (ICML), pp. 3929–3938, 2020.
- [11] Johnson, J., Douze, M., and Jégou, H. *Billion-scale Similarity Search with GPUs*. IEEE Transactions on Big Data, 7(3):535–547, 2019.

- [12] Khattab, O., and Zaharia, M. *ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT*. Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 39–48, 2020.
- [13] Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., Riedel, S., and Grave, E. *Atlas: Few-shot Learning with Retrieval Augmented Language Models*. Journal of Machine Learning Research, 24(251):1–43, 2023.
- [14] Ram, O., Levine, Y., Dalmedigos, I., Muhlgray, D., Shashua, A., Leyton-Brown, K., and Shoham, Y. *In-Context Retrieval-Augmented Language Models*. Transactions of the Association for Computational Linguistics, 11:1316–1331, 2023.
- [15] Borgeaud, S., Mensch, A., Hoffmann, J., Cai, T., Rutherford, E., Millican, K., van den Driessche, G. B., Lespiau, J.-B., Damoc, B., Clark, A., de Las Casas, D., Guy, A., Menick, J., Ring, R., Hennigan, T., Huang, S., Maggiore, L., Jones, C., Cassirer, A., Brock, A., Paganini, M., Irving, G., Vinyals, O., Osindero, S., Simonyan, K., Rae, J. W., Elsen, E., and Sifre, L. *Improving Language Models by Retrieving from Trillions of Tokens*. Proceedings of the 39th International Conference on Machine Learning (ICML), pp. 2206–2240, 2022.
- [16] Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W. W., Salakhutdinov, R., and Manning, C. D. *HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering*. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 2369–2380, 2018.
- [17] Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. *MTEB: Massive Text Embedding Benchmark*. Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics (EACL), pp. 2014–2037, 2023.
- [18] Sarthi, P., Abdullah, S., Tuli, A., Khanna, S., Goldie, A., and Manning, C. D. *RAPTOR: Recursive Abstractive Processing for Tree-Organized Retrieval*. International Conference on Learning Representations (ICLR), 2024.
- [19] Asai, A., Wu, Z., Wang, Y., Sil, A., and Hajishirzi, H. *Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection*. arXiv:2310.11511, 2023.
- [20] Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., Truitt, S., and Larson, J. *From Local to Global: A Graph RAG Approach to Query-Focused Summarization*. arXiv:2404.16130, 2024.
- [21] Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., and Wang, H. *Retrieval-Augmented Generation for Large Language Models: A Survey*. arXiv:2312.10997, 2024.
- [22] Gupta, S., Ranjan, R., and Singh, S. N. *A Comprehensive Survey of Retrieval-Augmented Generation (RAG): Evolution, Current Landscape and Future Directions*. arXiv:2410.12837, 2024.

- [23] Zhao, P., Zhang, H., Yu, Q., Wang, Z., Geng, Y., Fu, F., Yang, L., Zhang, W., Jiang, J., and Cui, B. *Retrieval-Augmented Generation for AI-Generated Content: A Survey*. arXiv:2402.19473, 2024.
- [24] Smith, B., and Troynikov, A. *Evaluating Chunking Strategies for Retrieval*. Chroma Technical Report, July 2024.
- [25] Yu, H., Gan, A., Zhang, K., Tong, S., Liu, Q., and Liu, Z. *Evaluation of Retrieval-Augmented Generation: A Survey*. arXiv:2405.07437, 2024.
- [26] Xiong, W., Liu, J., Molybog, I., Zhang, H., Bhargava, P., Hou, R., and Martin, L. *Effective Long-Context Scaling of Foundation Models*. arXiv:2309.16039, 2023.
- [27] Anthropic. *Introducing Claude 2.1*. Technical report, 2023. Available at: <https://www.anthropic.com/index/claude-2-1>
- [28] OpenAI. *GPT-4 Technical Report*. arXiv:2303.08774, 2023.
- [29] Nogueira, R., and Cho, K. *Passage Re-ranking with BERT*. arXiv:1901.04085, 2019.
- [30] Pirolli, P., and Card, S. *Information Foraging*. Psychological Review, 106(4):643–675, 1999.
- [31] Lipton, Z. C. *The Mythos of Model Interpretability: In Machine Learning, the Concept of Interpretability is Both Important and Slippery*. Queue, 16(3):31–57, 2018.
- [32] Doshi-Velez, F., and Kim, B. *Towards a Rigorous Science of Interpretable Machine Learning*. arXiv:1702.08608, 2017.