

# Cheat Sheet: Advanced Retrievers for RAG



Estimated Reading Time: 15 minutes

## Core Retrieval Concepts

### What are Advanced Retrievers?

Advanced retrievers go beyond simple vector similarity search to provide more nuanced, context-aware information retrieval through:

- **Semantic Understanding:** Using embeddings for meaning and context
- **Keyword Matching:** Precise term-based search for exact specifications
- **Hierarchical Context:** Maintaining relationships between information levels
- **Multi-Query Processing:** Generating and combining results from multiple query variations
- **Fusion Techniques:** Intelligently combining results from different retrieval methods

### Maximum Marginal Relevance (MMR)

**Purpose:** Balance relevance and diversity of retrieved results

**Method:** Selects documents that are highly relevant to the query AND minimally similar to previously selected documents

**Benefit:** Avoids redundancy and ensures comprehensive coverage of different query aspects

---

## LlamaIndex Retrievers

### Core Index Types in LlamaIndex

#### VectorStoreIndex

- **Function:** Stores vector embeddings for each document chunk
- **Best suited for:** Semantic retrieval based on meaning
- **Usage:** Commonly used in LLM pipelines and RAG applications

#### DocumentSummaryIndex

- **Function:** Generates and stores summaries of documents at indexing time
- **Process:** Uses summaries to find and retrieve relevant documents
- **Best for:** Large documents whose meanings would be lost by chunking; large documents that cannot fit in LLM or embedding model context windows
- **Key Points:** Returns original documents, not their summaries; uses summaries instead of text chunks to enable retrieval based on the semantic meaning of the entire text

#### KeywordTableIndex

- **Function:** Extracts keywords from documents and maps to content chunks
- **Best for:** Exact keyword matching for rule-based or hybrid search scenarios
- **Use Case:** Applications requiring precise term matching

### LlamaIndex Retriever Types

#### 1. Vector Index Retriever

**Most common retriever** - uses vector embeddings to find semantically related content

- **Process:** Embeds query, compares with document embeddings using cosine similarity
- **Ideal for:** General-purpose search, RAG pipelines where semantic understanding is crucial
- **Limitation:** May miss exact keyword matches when specific terms are crucial

#### 2. BM25 Retriever

**Advanced keyword-based retrieval** that improves on TF-IDF

##### TF-IDF Foundation:

- **Term Frequency (TF):** How often a word appears in a document
- **Inverse Document Frequency (IDF):** How rare a word is across all documents
- **TF-IDF Score:**  $TF \times IDF$

##### BM25 Improvements:

- **Term Frequency Saturation:** Reduces impact of repeated terms using saturation function
- **Document Length Normalization:** Adjusts for document length, preventing long document bias
- **Tunable Parameters:**  $k1 \approx 1.2$  (saturation control),  $b \approx 0.75$  (length normalization)

**Best for:** Technical documentation, legal documents, exact terminology requirements

#### 3. Document Summary Index Retrievers

Two Variants:

- 1. **DocumentSummaryIndexLLMRetriever**: Uses LLM to analyze query against summaries (intelligent but expensive)
- 2. **DocumentSummaryIndexEmbeddingRetriever**: Uses semantic similarity between query and summary embeddings (faster, cost-effective)

**Process:** Two-stage approach using summaries to filter documents, then returns full document content

4. Auto Merging Retriever

**Purpose:** Preserves context in long documents using hierarchical structure  
**Method:**

- Uses hierarchical chunking (parent and child nodes)
- If enough child nodes from same parent are retrieved, returns parent node instead
- **Dual Storage:** Child chunks for precise matching, parent chunks for context

**Best for:** Long documents, legal papers, technical specifications needing context preservation

5. Recursive Retriever

**Purpose:** Follows relationships between nodes using references  
**Capability:** Can follow references from one node to another (citations, metadata links)  
**Types:** Supports chunk references and metadata references  
**Best for:** Academic papers with citations, interconnected knowledge bases

6. Query Fusion Retriever

**Purpose:** Combines results from different retrievers and optionally generates multiple query variations

Core Capabilities:

- Multiple retriever support (combines vector-based and keyword-based methods)
- Query variation generation using LLM
- Sophisticated fusion strategies to improve recall

Three Fusion Modes:

Reciprocal Rank Fusion (RRF)

- **Most robust fusion method** - combines ranked lists using reciprocal of ranks
- **Formula:**  $RRF\_score(d) = \sum (1 / (rank\_i(d) + k))$  where  $k \approx 60$
- **Best for:** Default choice for most fusion scenarios, production systems

Relative Score Fusion

- **Preserves score magnitudes** while normalizing across query variations
- **Formula:**  $normalized\_score = original\_score / max\_score$
- **Best for:** When embedding model confidence scores are meaningful

Distribution-Based Score Fusion

- **Most sophisticated** - uses statistical properties of score distributions
- **Methods:** Z-score normalization, percentile ranking
- **Best for:** Complex queries with varying score distributions

LangChain Retrievers

LangChain Retriever Interface

**Definition:** "An interface that returns documents based on an unstructured query"

- More general than a vector store
- Accepts string query as input, returns list of documents as output
- Doesn't necessarily store documents - purpose is to retrieve them

LangChain Retriever Types

1. Vector Store-Backed Retriever

**Foundation retriever** - lightweight wrapper around vector store class  
**Search Types:**

- **Simple Similarity Search:** Returns documents ranked by similarity (default 4 results)
- **MMR Search:** Balances relevance and diversity to avoid redundancy
- **Similarity Score Threshold:** Returns only documents above specified threshold

2. Multi-Query Retriever

**Problem Addressed:** "Distance-based vector database retrieval may vary with subtle changes in query wording"

Solution Process:

- 1. Uses LLM to generate multiple queries from different perspectives

2. **For each query, retrieves set of relevant documents**
3. **Takes unique union** of results for larger set of potentially relevant documents

**Benefit:** "By generating multiple perspectives on the same question, the MultiQueryRetriever can potentially overcome some limitations of distance-based retrieval"

### 3. Self-Querying Retriever

**Core Capability:** "Has the ability to query itself"

**Process:** Converts natural language query into structured query with two components:

1. **String to look up semantically**
2. **Metadata filter** to accompany it

**Requirements:** Documents must have rich, structured metadata with field descriptions

**Best for:** Applications combining semantic search with attribute filtering

**Example Queries:**

- "I want to watch a movie rated higher than 8.5" (filter only)
- "Has Greta Gerwig directed any movies about women" (query + filter)

### 4. Parent Document Retriever

**Problem Solved:** "Conflicting desires" when splitting documents:

- Small documents for accurate embeddings
- Large documents for context retention

**Solution:** "Strikes that balance by splitting and storing small chunks of data"

**Process:**

1. During retrieval, first fetches small chunks
2. Looks up parent IDs for those chunks
3. Returns larger documents containing the small chunks

**Architecture:**

- **Two splitters:** Parent (large chunks for retrieval) and child (small chunks for embeddings)
- **Dual storage:** Vector store for embeddings, document store for parent documents

## Decision Framework

| Need                    | LlamaIndex Choice                                  | LangChain Choice                           |
|-------------------------|----------------------------------------------------|--------------------------------------------|
| Exact keyword matching  | BM25 Retriever                                     | Vector Store-Backed + custom keyword logic |
| Multi-query with fusion | Query Fusion Retriever (RRF/Relative/Distribution) | Multi-Query Retriever (union approach)     |
| Citation following      | Recursive Retriever                                | Not directly supported                     |
| Hierarchical context    | Auto Merging Retriever                             | Parent Document Retriever                  |
| Simple semantic search  | Vector Index Retriever                             | Vector Store-Backed Retriever              |

## Author(s)

[Wojciech "Victor" Fulmyk](#)



**Skills Network**