1

Information Retrieval

In the first chapter of this book, we are going to learn the fundamentals of information retrieval (IR) systems. We are going to dive into details of the definition of IR, its core components, and major applications in **search** and **recommendation** systems, which establish the foundation for subsequent chapters by focusing exclusively on these two application domains.

1.1 Definition and Historical Evolution

IR broadly refers to the process of obtaining relevant information to meet a user's needs. While its essence remains constant—connecting people with the information they seek—its scope and meaning have expanded significantly over time, reflecting the evolution of technology and the growing complexity of information systems.

In this section, we will explore its foundational definition and trace its evolution to build a comprehensive understanding of the IR landscape.

1.1.1 Definition

IR is the science and practice of obtaining relevant information resources that satisfy a user's information needs from a large collection of data. As illustrated in Figure 1.1, IR systems serve as the bridge between users and the vast pool of information, facilitating the retrieval of relevant responses based on user inputs or inferred preferences.

The diagram shows the basic workflow of an IR system, where a user's request (implicit or explicit) is processed to retrieve relevant documents, and these documents are then transformed into meaningful responses. This simplified illustration highlights the core components:

- **User Request**: Represents the user's input or inferred need. This could be an explicit query, such as a search phrase or natural language question, or an implicit signal, such as browsing behavior or preferences.
- **Documents**: The source of information, which could include structured data (e.g., product catalogs, relational databases), semi-structured data (e.g., XML, JSON), or unstructured data (e.g., articles, reviews, multimedia). The IR system organizes and stores this data in formats that allow for efficient search and retrieval.
- **Retrieval Engine**: At the heart of the IR process is the retrieval engine, which matches the user request against a vast collection of data to find relevant information. This step relies on

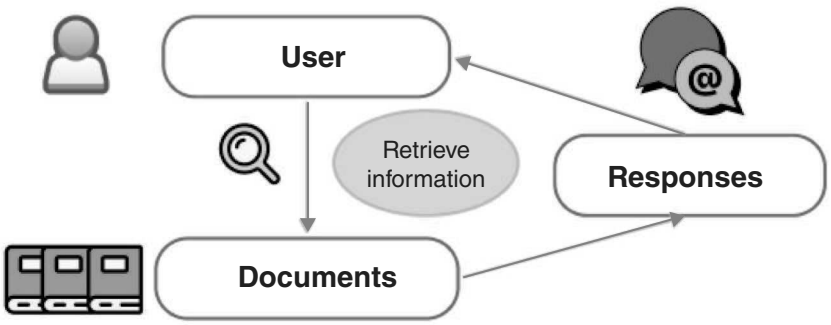


Figure 1.1 Information retrieval systems overview.

a combination of indexing, matching algorithms, and ranking mechanisms to identify and prioritize results.

- **Responses**: The system delivers the retrieved information in a form that meets the user's needs. Responses can take various formats, such as ranked lists of documents, summaries, or curated suggestions, depending on the application.

This concept of IR has evolved significantly over time, from traditional library systems to recent search engines and personalized recommendation systems. The primary purpose of IR is to deliver the right information at the right time, catering to user queries or inferred needs.

1.1.2 Historical Evolution

IR began as a manual process in libraries and archives, transitioning through several key phases:

- Manual Retrieval (pre-1950s): Catalog systems and index cards in libraries.
- Early Days and Text-based Era (1950s–1980s): Introduction of early retrieval systems, including Boolean search systems, vector space models, and probabilistic approaches.
- Web-era Information Retrieval (1990s–2000s): Emergence of link analysis and large-scale distributed systems.
- Neural IR (2010s–present): Integration of deep learning and contextual understanding.

As shown in Table 1.1, each era brought significant improvements in search effectiveness.

1.1.2.1 Manual Retrieval (Pre-1950s)

Libraries and archives relied on catalog systems and index cards to help users locate resources. **Cataloging**, which involves creating records and descriptions of the items in a collection to identify and access them, laid the foundation for today's IR systems by giving structure and findability to data.

Librarians played a central role, manually matching queries with relevant materials. In ancient Greek times, the Library of Alexandria used an early form of card catalogs that allowed scholars to search the scrolls and find important works of the day in this prominent center of learning and knowledge. Figure 1.2 illustrates a card catalog in a library where users can search for books by flipping through drawers of cards categorized by title, author, and subject.

Table 1.1 Evolution of search effectiveness.

Era	Search time	Accuracy (%)	Scale	Understanding level
Manual retrieval (pre-1950s)	Hours to days	10–15	Hundreds	Human-based classification
Early computerization (1950s–1960s)	Minutes to hours	20–30	Thousands	Boolean logic only
Text-based era (1970s–1980s)	Seconds to minutes	40–50	Millions	Term frequency based
Web era (1990s–2000s)	Sub-second	60–70	Billions	Link-based relevance
Neural era (2010s–present)	Milliseconds	80–90	Trillions	Semantic and contextual

Figure 1.2 Woman at main reading room card catalog, Library of Congress, circa 1930s [1].

1.1.2.2 Early Computerization (1950s–1980s)

The foundation of IR as a field was laid in the 1950s when computer scientist Calvin Mooers first coined the term "information retrieval" in his paper published in the *Proceedings of the International Congress of Mathematicians* [2]. This period marked the transition from manual indexing and searching methods to the earliest forms of automated retrieval systems.

1.1.2.2.1 The Birth of Boolean Search

One of the earliest and simplest approaches to data modeling in IR is based on set theory, where both documents and queries are treated as collections (or "buckets") of words. For example, Figure 1.3 illustrates the intersection of two sets A and B.

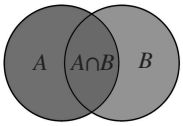
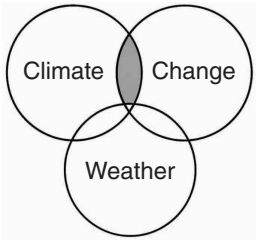


Figure 1.3 A set theory Boolean AND query [3].



climate AND change NOT weather

Figure 1.4 Boolean search on "climate AND change NOT weather."

This foundational approach gives rise to the use of **Boolean search systems**. These systems introduced three fundamental logical operators:

- AND requiring all specified terms to be present.
- OR requiring at least one of the specified terms.
- NOT excluding documents with certain terms.

These systems allowed users to construct queries with precision, targeting documents that matched specific criteria. For example, as illustrated in Figure 1.4, a query such as "climate AND change NOT weather" would return documents containing both "climate" and "change" while excluding those containing "weather."

1.1.2.2.2 Fuzzy Set Theory for Relevance Ranking

While traditional Boolean models operate on a strict match/no-match basis, **fuzzy set theory** refines this approach to allow for degrees of membership. In the context of IR, fuzzy logic introduces the concept of weighted relevance, enabling systems to rank documents based on how closely they match a query rather than just whether they match.

Key features of fuzzy set theory in IR include the following:

- Partial Matches: Documents are assigned a relevance score based on their degree of match with the query, rather than being strictly included or excluded.
- Relevance Ranking: This scoring allows documents to be sorted by relevance, improving the user experience by presenting the most relevant results first.

1.1.2.2.3 Vector Space Models

In the 1960s, Gerard Salton at Cornell University developed the "SMART system" [4], which was one of the earliest experimental systems to introduce advanced techniques that later became the cornerstone of modern IR. One pioneering concept proposed in the SMART system is the "vector space model," introducing a mathematical framework for representing documents and queries as vectors in a multidimensional space. This model was a major step forward in IR, moving beyond simple keyword matching to capture the relative importance of terms, significantly improving retrieval precision and ranking.

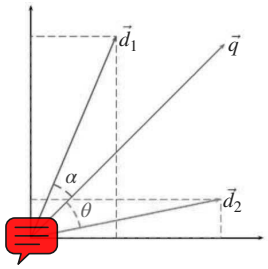


Figure 1.5 Cosine similarity between query and document vectors.

Each document is represented as a vector, where dimensions correspond to unique terms, and values represent their weights. Similarly, queries are also represented as vectors in the same space. The relevance of a document to a query is determined by calculating the cosine similarity between their respective vectors. A higher cosine similarity score indicates greater alignment and relevance. For example, a query such as "machine learning algorithms" would rank documents by how closely their vectors align with the query vector.

Figure 1.5 demonstrates this concept with a concrete example in a two-dimensional (2D) vector space. Consider a query q about "AI algorithms" and two documents: d_1 containing "machine learning algorithms" and d_2 containing "graph algorithms." The vectors represent their positions in the term space, where the angle between vectors determines their similarity. Document d_1 has a smaller angle with the query vector, indicating higher relevance and better alignment with the user's information need, while d_2 shows a larger angle, suggesting lower relevance.

1.1.2.2.4 Term Weights and the TF-IDF Examples

In vector space models, each term in a document is assigned a weight, which quantifies its importance within the document and across the entire collection. These weights form the components of the vector representation of a document. Term weights play a crucial role in determining the similarity between a query and a document, directly influencing the ranking of results.

Term frequencies and document frequencies are commonly used terms for the term weighting. Each term in a document is assigned a weight based on both:

- Term Frequency (TF)—how often the term appears in a document.
- Inverse Document Frequency (IDF)—how rare the term is across the corpus.

The **term frequency-inverse document frequency (TF-IDF)** score for a term t in document d is computed as:

$$\text{TF-IDF(t, d)} = \text{TF(t, d)} \times \text{IDF(t)}$$

To illustrate this in action, suppose we have a query "machine learning algorithms" and three candidate documents. For each document, we compute TF-IDF for the terms, construct vectors, and then rank the documents by their cosine similarity to the query vector. This ranks documents by relevance, which is the fundamental goal of IR: to order results based on how well they satisfy the user's intent. While here we briefly introduce the definition of TF-IDF, we are going to dive into more details of TF-IDF in Section 1.2.3 of the book.

Term weights are the cornerstone of the vector space model, providing the quantitative foundation for measuring the relevance of documents to user queries. These weighting schemes have evolved significantly, leading to more accurate and intuitive IR systems.

1.1.2.2.5 Probabilistic Models and BM25

The probabilistic approach to IR represents a sophisticated evolution beyond vector space models, focusing on estimating the probability that a document is relevant to a given query. Unlike deterministic approaches such as TF-IDF, probabilistic models explicitly handle uncertainty in the retrieval process through statistical inference.

At its core, the probabilistic model attempts to answer a fundamental question: given a user query Q, what is the probability $P(R|D)$ that a document D is relevant (R)? This framework is based on the probability ranking principle (PRP), which states that documents should be ranked according to their probability of relevance to the query.

The **binary independence model (BIM)** serves as a foundational example. It makes several key assumptions:

- Term Independence: The presence/absence of one term doesn't affect another.
- Binary Weighting: Terms are either present or absent.
- Document relevance is independent of other documents.

One of the most successful instantiations of probabilistic ranking is **Best Match 25 (BM25)**, which improves upon TF-IDF by introducing term saturation and length normalization.

BM25 refines relevance estimation by considering how often terms appear (TF), how informative they are (IDF), and how long the document is. As a probabilistic model, it remains one of the most widely used scoring functions in modern retrieval engines. While we briefly introduce the definition of BM25 here, we will discuss it in more details of BM25 in Section 1.2.3 of the book.

A more sophisticated approach is the **Bayesian network model**, which represents documents and queries as nodes in a directed acyclic graph (DAG). As illustrated in Figure 1.6, the network captures: (1) document nodes representing individual documents, (2) term nodes representing index terms, (3) query nodes representing user information needs, and (4) edges representing probabilistic dependencies between nodes.

In Bayesian network models, the relevance of a document is inferred through probabilistic dependencies between query terms, documents, and latent relevance variables. Given a query Q and document D, the system estimates:

$$P(R = 1 \mid Q, D)$$

by propagating beliefs through a graphical model encoding term–document relationships.

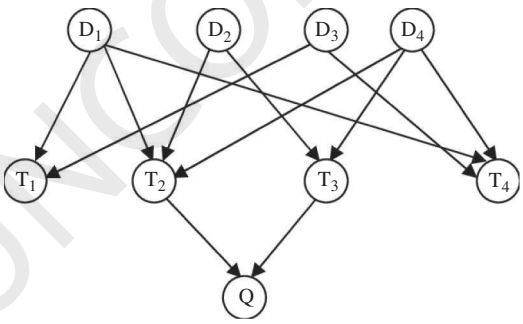


Figure 1.6 DAG for Bayesian network model. *D*s represent documents; *T*s represent terms; *Q* represents query; Edges represent probabilistic dependencies between nodes.

As machine learning (ML) continues to advance, probabilistic approaches remain fundamental to modern IR systems, often working in conjunction with neural networks and other AI techniques.

1.1.2.3 Web-era Information Retrieval (1990s–2000s)

The 1990s marked a transformative era for IR with the advent of the World Wide Web (WWW) and the emergence of early search engines. The web revolutionized how individuals accessed and shared information, enabling global-scale connectivity among users, organizations, and machines. This explosive growth in content volume and diversity led to a pressing need for scalable, efficient retrieval systems capable of delivering relevant information from billions of web documents—far beyond the scale of traditional IR systems.

1.1.2.3.1 Web Search as a Retrieval Paradigm

Web search evolved as a distinct and large-scale retrieval method, building upon core IR principles while introducing new architectural and algorithmic innovations. Compared to traditional IR, web search differs in several ways:

- Scale: Web collections include billions of heterogeneous documents spanning text, images, and structured data.
- Noise and Adversarial Content: Web content includes spam, duplicated pages, and low-quality or misleading documents.
- User Interaction: Queries tend to be short (2–3 keywords), with users expecting instant and high-quality results.
- Architecture: Large-scale distributed infrastructure is essential to support indexing, crawling, ranking, and query response.

While web search inherits foundational concepts from vector space models, probabilistic retrieval, and Boolean logic, it diverges significantly in implementation due to its scale and complexity. As illustrated in Figure 1.7, it introduced new components such as web crawling, link analysis, distributed indexing, and real-time ranking.

1.1.2.3.2 Early Search Engines and System Architecture

In the early days of the Web, search engines like Yahoo! initially relied on human-curated directories to organize content (Figure 1.8). These hierarchical listings worked well for a small and structured web but quickly became inadequate as the number of websites exploded.

To address this, search engines began building automated indexing systems powered by web crawlers and distributed processing pipelines. Pioneers like AltaVista were among the first to offer full-text search over large-scale web documents, leveraging parallel architectures for crawling, storage, and retrieval. These systems laid the groundwork for modern search infrastructure, which partitions the web across data centers, supports real-time updates, and handles millions of concurrent queries.

1.1.2.3.3 PageRank and Link-based Retrieval

A major advancement in web IR came in the late 1990s when Larry Page and Sergey Brin at Stanford introduced PageRank, a link-based ranking algorithm that measured the relative importance of web pages based on their hyperlink structure.

Unlike traditional keyword-based scoring methods, PageRank treated hyperlinks as votes of confidence. The core idea was that a page is important if it is linked to by other important pages. This approach was inspired by academic citation analysis. Figure 1.9 illustrates this with two example

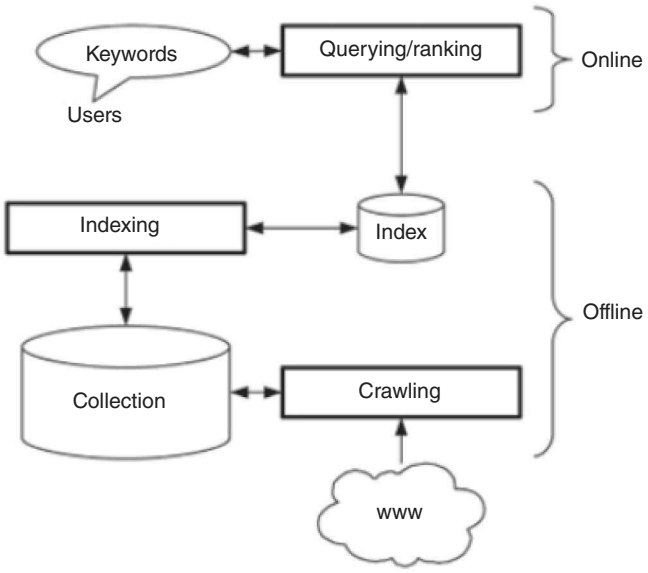


Figure 1.7 Phases of web search [5].

Figure 1.8 Yahoo search page in 2005.

nodes: Node B (a highly cited paper) receives many incoming links and is considered authoritative; Node F (a rarely cited paper) has only a few links and is considered less important.

Thus, PageRank captures the notion of authority propagation across the web graph. The algorithm computes a score for each page using the structure of the graph, assigning higher scores to pages with more influential incoming links.

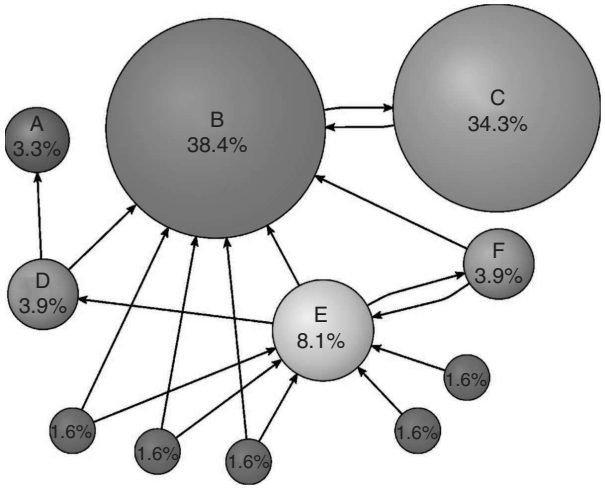


Figure 1.9 Simplified illustration of PageRank with citation-based analogy. Percentages represent perceived importance; arrows represent hyperlinks [6].

While PageRank represented an important innovation, it was only one part of Google's success story on search engines (Figure 1.10). Other crucial factors included:

- Comprehensive Indexing: Google's aggressive web crawling and indexing strategy captured a larger portion of the web than competitors, providing more comprehensive search results.
- System Performance: Superior infrastructure and algorithms enabled faster query processing and more responsive user experiences.
- Algorithm Integration: PageRank was combined with traditional IR techniques and other ranking factors to create a more robust relevance assessment.
- Continuous Innovation: Google consistently refined its algorithms, incorporating user behavior signals, content quality metrics, and contextual factors.

These complementary components together contributed to Google's ability to consistently deliver relevant, high-quality results, helping it surpass competitors like Yahoo and AltaVista.

1.1.2.3.4 Bridging Web Search and Traditional IR

Despite their architectural differences, web search systems are deeply rooted in traditional IR frameworks. For example, vector space models are still used for scoring document-query relevance; probabilistic models, like BM25, remain foundational in ranking functions; Boolean filters are applied to support structured constraints (e.g., site: filters, date ranges); link analysis (e.g., PageRank) added a graph-based dimension to IR, enriching the representation and evaluation of document authority; user interaction data, such as click-through rates (CTRs) and dwell time, further extend the retrieval model by integrating behavioral signals.

Thus, web search represents an evolutionary integration of existing IR methods with new techniques designed for web-scale data, heterogeneous content, and real-time responsiveness.

Figure 1.10 Google search page in 2001.

1.1.2.4 Big Data and ML (2000s–present)

The early twenty-first century ushered in the big data era, characterized by a massive influx of structured and unstructured data from sources such as social media, mobile devices, e-commerce platforms, Internet of Things (IoT) sensors, and cloud services. This explosion of data volume, velocity, and variety exposed the limitations of traditional keyword-based retrieval approaches, which were often brittle, syntactic, and context-insensitive.

To meet these new demands, IR systems evolved to become artificial intelligence (AI)-powered, leveraging ML, deep learning (DL), and natural language processing (NLP) to understand user intent, rank results by relevance, and deliver personalized responses. These technologies are key components within the broader field of AI, as illustrated in Figure 1.11, which shows the conceptual relationship between these fields:

- AI: The broadest category, encompassing all techniques that enable machines to mimic human intelligence, including reasoning, perception, and decision-making.
- ML: A subset of AI focused on algorithms that learn patterns from data and improve over time without being explicitly programmed.
- DL: A specialized area of ML that uses artificial neural networks with multiple layers to model complex patterns in high-dimensional data.
- NLP: An application domain of AI and ML focused on understanding, generating, and interacting with human language.

1.1.2.4.1 Machine Learning

ML enables systems to learn from user interactions, past behavior, and content characteristics to improve search relevance, recommendation quality, and query understanding. Traditional IR systems used manually engineered features and fixed rules. In contrast, ML-based systems automatically discover useful features, adapt to new data, and optimize ranking functions through training.

One commonly used family of algorithms in IR is learning-to-rank, which uses labeled relevance data to train models that rank documents given a query. These models outperform traditional TF-IDF and BM25 methods in many real-world settings.

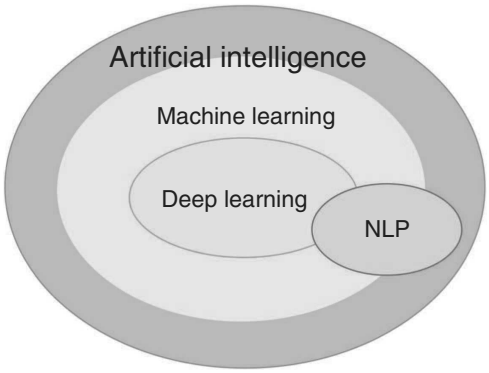


Figure 1.11 Relationships among AI, ML, DL, and NLP (Note: NLP intersects with AI, ML, and DL—it is an application domain that uses techniques from all three, not a subset of any single field).

1.1.2.4.2 Deep Learning

DL has significantly advanced IR by allowing systems to learn hierarchical representations from raw inputs such as text, audio, and images. Unlike shallow models that rely on manual feature extraction, DL models learn distributed, abstract representations that capture semantic meaning.

Two foundational architectures in DL are:

- Convolutional Neural Networks (CNNs): Originally developed for image recognition, CNNs have also been applied to text classification and matching tasks in IR. They work by learning local patterns via convolutional filters and aggregating them for global understanding.
- Recurrent Neural Networks (RNNs): Designed for sequential data, RNNs are used in tasks such as query suggestion, click prediction, and document summarization. They maintain a memory of past inputs, making them suitable for modeling word sequences and capturing context over time. Variants such as long short-term memory (LSTM) and gated recurrent units (GRU) address the limitations of standard RNNs in long-sequence learning.

These models enabled neural IR systems that go beyond bag-of-words and TF, allowing systems to rank based on semantic similarity, not just lexical overlap.

1.1.2.4.3 Natural Language Processing

NLP applies ML and DL to understand and generate human language. Historically, NLP originated from rule-based symbolic AI and only later incorporated statistical and DL methods. For example, early systems such as ELIZA [7], a pattern-matching chatbot that simulated a psychotherapist, and SHRDLU [8], which understood natural language commands within a blocks-world environment, relied entirely on hand-crafted rules and symbolic reasoning rather than statistical learning. Thus, NLP is best viewed as an overlapping domain, not a subset of ML.

In IR, some examples of how NLP techniques can help systems: (1) parse user queries to identify intent and disambiguate meaning; (2) extract entities and relations; (3) match queries to semantically related documents; (4) generate readable answers or summaries. For example, a traditional search for "date" might return ambiguous results related to fruit, calendar dates, or social interactions. NLP-powered systems analyze surrounding context (e.g., in "best date ideas"), detect that the user likely means romantic outings, and retrieve accordingly. We are going to introduce more details of the NLP tasks in Section 2.1.1.2.

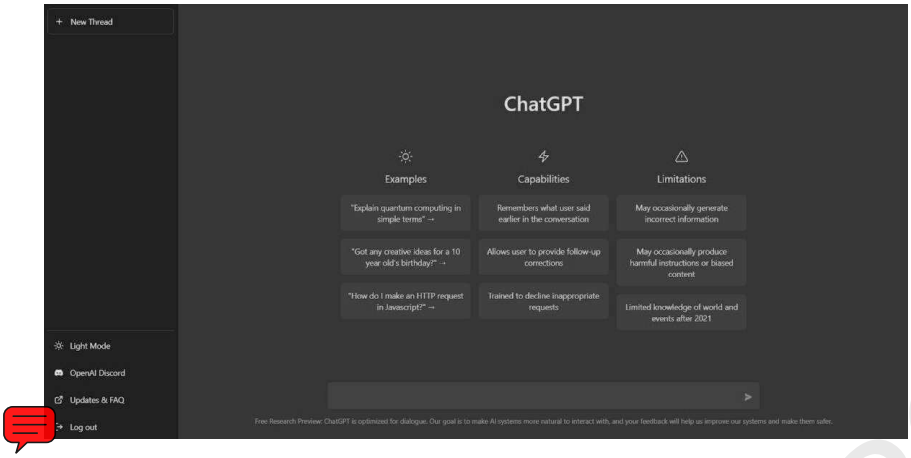


Figure 1.12 OpenAI's ChatGPT UI in 2023.

1.1.2.4.4 Toward Conversational and Personalized IR

Modern AI-powered IR systems consider user preferences, behavioral signals, and contextual information to tailor results for each individual. E-commerce platforms use these techniques for recommendation systems, suggesting relevant products based on previous behavior, clicks, and natural language queries.

In parallel, conversational AI has emerged as a natural evolution of retrieval technology. Systems now support interactive dialog, allowing users to refine queries, ask follow-up questions, and receive dynamic, multi-turn responses.

One prominent example is OpenAI's GPT (generative pre-trained transformer) family, including ChatGPT (Figure 1.12), which can interpret and respond to user input in a human-like, context-aware manner. These large language models (LLMs) bring together DL, massive data, and transformer architectures to push the boundaries of intelligent search and retrieval. From personalizing e-commerce recommendations to answering complex questions, these innovations redefine how we access and interact with information in a data-rich world.

1.2 Information Retrieval Components

1.2.1 Overview

Modern IR systems have evolved beyond the basic elements shown in Figure 1.1 (users, documents, and responses) into a sophisticated, systematic framework. There are essential components that form the backbone of contemporary IR systems: document indexing, user input processing, retrieval, ranking, and evaluation. As shown in Figure 1.13, these components can be categorized into three different stages: "pre-retrieval," "retrieval," and "post-retrieval."

1.2.1.1 User Input Handling

Modern IR systems account for diverse user inputs, including explicit queries and implicit signals such as browsing history or contextual interactions. This component involves two key tasks:

- Query Understanding: Interpreting and refining explicit user queries (e.g., keywords) for better alignment with document representations.

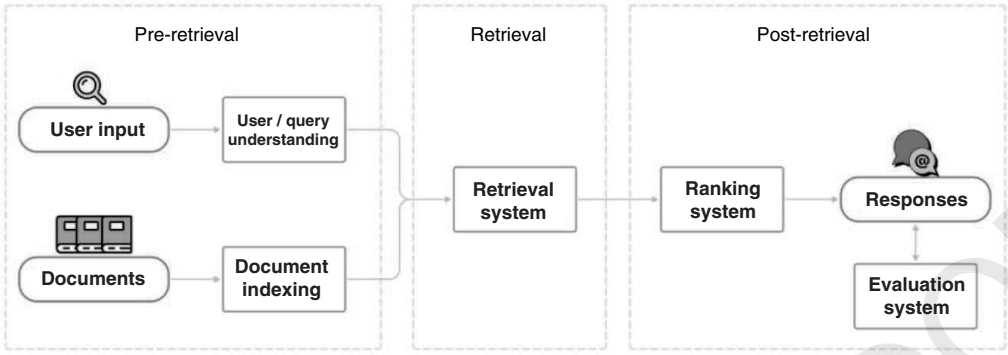


Figure 1.13 Pre-retrieval, retrieval and post-retrieval.

- User Understanding: Incorporating user behavior (e.g., browsing history), preferences, and context to personalize results, particularly for recommendations.

1.2.1.2 Document Handling

This is the process of transforming raw documents into a structured format that facilitates efficient retrieval. Indexing organizes content in a way that enables quick and accurate matching with both explicit queries and implicit signals. This foundational step creates the data structures necessary for efficient retrieval operations.

1.2.1.3 Retrieval

Based on the processed user inputs—whether explicit queries or implicit signals—the system retrieves relevant documents from the indexed repository. A dual retrieval architecture is used to ensure comprehensive coverage of relevant documents:

- Sparse Retrieval: Efficiently handles traditional keyword-based matching.
- Dense Retrieval: Enables semantic understanding and matching.

1.2.1.4 Ranking

Once candidate documents are retrieved, they are ranked based on their relevance to the user input. The ranking system includes:

- Ranker: Initial ranking based on relevance signals such as content similarity and query-document matches.
- Re-ranker: A secondary, more sophisticated ranking layer that refines the order using advanced techniques such as ML models. The initial ranker performs coarse screening based on quickly calculated signals (such as keyword matching), while the re-ranker uses more computationally expensive models (such as neural networks) for fine sorting.

1.2.1.5 Evaluation

The effectiveness of the IR system is assessed through metrics such as precision, recall, and user satisfaction. For implicit inputs, additional metrics, such as CTR or engagement duration, are used to measure system performance and refine its recommendations. For example, if a recommended product has a lower CTR than average, the system will lower its priority in similar queries.

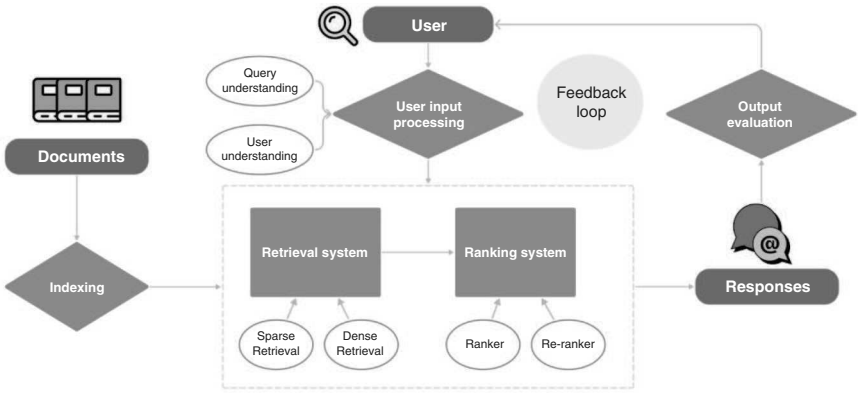


Figure 1.14 Key components of modern IR systems.

The diagram in Figure 1.14 shows how these components form an integrated pipeline, with the feedback loop creating a dynamic system that learns and improves from user interactions. Documents flow through indexing to the retrieval-ranking subsystem (highlighted in the dotted box), while user input undergoes sophisticated processing before entering this core subsystem. The final responses are evaluated, completing the feedback cycle.

This systematic approach enables modern IR systems to cater to diverse use cases, from search engines to personalized recommendation platforms, by seamlessly integrating user signals, retrieval mechanisms, and adaptive evaluation techniques. In the next sections, we are going to dive deeply into each component.

1.2.2 Pre-retrieval

The pre-retrieval phase in modern IR systems consists of two critical components: user input handling and document handling. These two components operate independently: document processing prepares the corpus **offline**, while user input processing runs **online** to adapt to real-time queries. These components lay the groundwork for effective IR by preparing both the query and document corpus for optimal matching.

1.2.2.1 User Input Handling

User input handling transforms raw user queries or implicit signals into a format that can be effectively matched against indexed documents. This process involves several sophisticated steps:

1.2.2.1.1 Query Processing

At the core of user input handling is query processing, which aims to enhance the query's structure, clarity, and expressiveness. The goal is to bridge the gap between what the user types and what they actually mean, increasing the likelihood of retrieving relevant documents. This includes:

- Intent Classification: Determining the type of query (informational, navigational, or transactional) to better align retrieval strategies.
- Query Parsing: Breaking down the input query into meaningful components, identifying key terms, phrases, and potential entities.
- Query Expansion: Enriching the original query with synonyms, related terms, or contextual information to improve recall. Imagine searching for "iPhone charger": query expansion

might add "apple lightning cable" to broaden results, while contextual processing could prioritize sellers near your current location.
- Query Normalization: Normalizing and reformulating problematic queries to improve their likelihood of matching relevant documents. For example, correcting misspellings ("reciepe" → "recipe") or resolving ambiguous terms ("apple" → "Apple Inc.") to improve relevance.

1.2.2.1.2 Contextual Processing

In addition to query refinement, contextual processing seeks to personalize and ground the query using surrounding signals. This makes the system more adaptive to individual users and their situations. It includes:

- Session Context: Incorporating information from the user's current search session to better understand their immediate needs.
- Historical Context: Using the user's past interactions and preferences to personalize query interpretation.
- Environmental Context: Considering factors like location, time, and device type to provide more relevant results.

We are going to cover more details of query processing and contextual processing in Section 1.3 for the search and recommendations system applications.

1.2.2.2 Document Handling

While user input processing adapts queries in real time, document handling forms the offline backbone of IR systems. This component prepares the corpus for fast and accurate retrieval by converting raw documents into structured representations. The two primary steps involved are document preprocessing and index construction. Together, they transform unstructured content into efficient data structures that support millisecond-level query response times in production search and recommendation systems.

1.2.2.2.1 Document Preprocessing

Before any index can be built, documents must first be normalized and cleaned. Document preprocessing consists of a series of text normalization techniques that reduce noise and standardize content, thereby improving both retrieval speed and accuracy. These operations ensure that semantically relevant terms are retained while redundant or irrelevant information is filtered out.

- **Stopword Removal** eliminates functional words that carry little semantic meaning, such as articles ("the," "a"), prepositions ("in," "on"), and common verbs ("is," "are"). This process reduces index size by 20–40% while often improving retrieval precision by focusing on content-bearing terms.
- **Stemming** extracts morphological roots from words, allowing retrieval systems to match related word forms. A search for "housing" can retrieve documents containing "house," "houses," or "housing," significantly improving recall without manual query expansion.
- **Tokenization** separates continuous text into discrete word units. While straightforward for space-separated Western languages, this process becomes significantly more complex for languages like Chinese, Japanese, or Arabic, where word boundaries are not explicitly marked or where morphological complexity requires sophisticated analysis. We are going to discuss more details of tokenization in a language model context in Section 2.1.2.

1.2.2.2.2 Index Construction

Once documents are preprocessed, the next step is to build indexes—the core data structures that make fast and scalable retrieval possible. Without indexing, searching through massive corpora would be prohibitively slow. Index construction converts the preprocessed documents into optimized structures that support efficient lookups and sophisticated ranking algorithms.

The indexing method consists of several key operations:

- Inverted Index Creation: Maps terms to their document locations, forming the backbone of fast term-based search.
- Forward Index Creation: Maps documents to term features, useful for ranking, summarization, or filtering.
- Positional Information: Stores word positions within documents to support phrase and proximity searches.
- Metadata Integration: Incorporates additional document attributes (e.g., author, date, type) into the index to enable faceted search or filters.

An **inverted index** is the dominant data structure in traditional search engines. It allows constant-time access to all documents containing a given term. An inverted index consists of two primary components:

- Vocabulary (Dictionary): A sorted list of all unique terms appearing in the document collection, serving as the primary access point for retrieval operations.
- Posting Lists: For each vocabulary term, a list containing references to all documents where that term appears. Posting lists may also store:
  - TF in each document
  - Term position for phrase matching
  - Document frequency (DF) for ranking algorithms like TF-IDF

There are advanced indexing techniques, including:

- Positional Information: For phrase queries or proximity searches, indexes must store the exact positions where terms appear within documents, significantly increasing storage requirements but enabling sophisticated query types.
- Distributed Indexing: Large-scale systems achieve high performance through parallelization, requiring index partitioning strategies. Two primary approaches exist: document partitioning (each machine handles a subset of documents) and term partitioning (each machine handles a subset of vocabulary terms).
- Compression Techniques: Modern indexes employ various compression methods to reduce storage requirements while maintaining query performance, including delta compression for posting lists and front-coding for vocabulary storage.

These indexing innovations are designed to meet a series of critical system-level requirements. Scalability ensures that the indexing infrastructure can accommodate growing datasets without compromising performance. Latency must be minimized to support real-time applications, such as autocomplete or web search, where users expect sub-second responses. The system must also be flexible, able to handle various document formats and query types—from short keywords to long natural language questions. Just as important is maintainability: indexes must support updates, deletions, and rebuilds with minimal disruption. Lastly, resource efficiency is paramount. Index construction and retrieval should make optimal use of CPU, memory, and storage to reduce operational costs and energy consumption.

1.2.2.2.3 Example

Here we use an example to demonstrate the document processing and indexing workflow. Let's say we have a list of AI research papers consisting of six documents:

```
Document 1: "Machine learning algorithms improve natural language processing tasks"
Document 2: "Deep learning revolutionizes computer vision and image processing"
Document 3: "Natural language models enhance machine translation accuracy"
Document 4: "Computer vision algorithms detect objects in real-time processing"
Document 5: "Machine learning applications span computer science and data processing"
Document 6: "Deep neural networks advance artificial intelligence and learning
             systems"
```

The process of constructing the index is as follows: Extract Terms → Build Vocabulary → Create Posting Lists → Sort & Index

Step 1: Document Preprocessing: Before building an index, documents are tokenized and converted into a "bag of words" (BoW) representation: an unordered collection of terms, often paired with frequency counts. Each document is converted into a dictionary of term frequencies. Figure 1.15 shows the BoW results of each document, where each document is represented as an unordered collection of words, with frequency counts but no positional information.

Step 2: Document Indexing: The resulting inverted index is listed in Table 1.2.

Step 3: Boolean Query Using Constructed Index: Inverted indexes make query evaluation efficient via set operations. For example, for the Boolean query: "learning AND computer," here are the steps to use inverted index to retrieve documents:

1. Term Lookup:
   - Retrieve posting list for "learning": {1, 2, 5, 6}
   - Retrieve posting list for "computer": {2, 4, 5}
2. Set Intersection
   - Compute intersection: $\{1, 2, 5, 6\} \cap \{2, 4, 5\} = \{2, 5\}$
3. Result Interpretation
   - Document 2: "Deep **learning** revolutionizes **computer** vision and image processing."
   - Document 5: "Machine **learning** applications span **computer** science and data processing."

This example demonstrates how Boolean queries are efficiently resolved through mathematical set operations, enabling rapid retrieval even from large document collections.

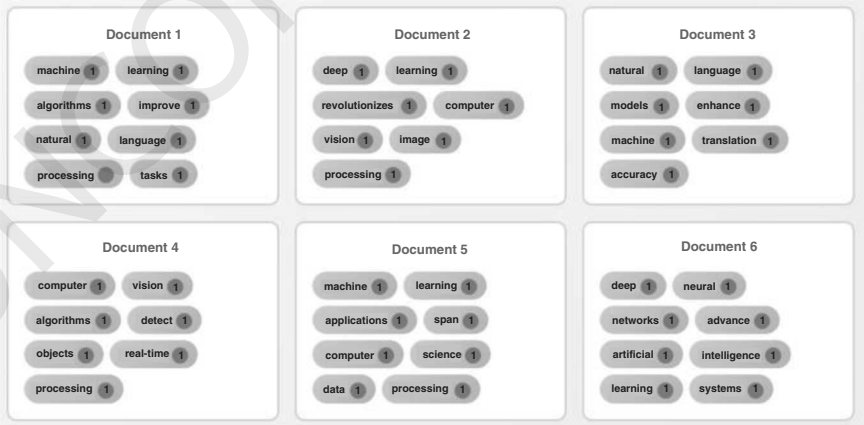


Figure 1.15 Bag of words representation.

Table 1.2 Inverted index table.

Term (vocabulary)	Posting list (Document IDs)	Document frequency
Accuracy	[9]	1
Advance	[10]	1
Algorithms	[5, 11]	2
Applications	[12]	1
Artificial	[10]	1
Computer	[11–13]	3
Data	[12]	1
Deep	[10, 13]	2
Detect	[11]	1
Enhance	[9]	1
Image	[13]	1
Improve	[5]	1
Intelligence	[10]	1
Language	[5, 9]	2
Learning	[5, 10, 12, 13]	4
Machine	[5, 9, 12]	3
Models	[9]	1
Natural	[5, 9]	2
Networks	[10]	1
Neural	[10]	1
Objects	[11]	1
Processing	[5, 11–13]	4
Real time	[11]	1
Revolutionizes	[13]	1
Science	[12]	1
Span	[12]	1
Systems	[10]	1
Tasks	[5]	1
Translation	[9]	1
Vision	[11, 13]	2

The pre-retrieval phase is crucial for system performance as it directly impacts both retrieval accuracy and efficiency. Well-processed user inputs and optimized document indexes create the foundation for sophisticated matching and ranking in subsequent phases.

1.2.3 Retrieval Models

IR systems employ a dual retrieval architecture that combines the efficiency of sparse retrieval with the semantic understanding capabilities of dense retrieval. This section explores both approaches and their integration in contemporary IR systems.

Table 1.3 lists the comparisons of sparse retrieval and dense retrieval. The combination of sparse and dense retrieval techniques in modern search systems allows for a comprehensive approach to document retrieval. Sparse methods excel at precise keyword matching and handling straightforward queries, while dense methods capture semantic nuances and handle more complex, context-dependent searches. This dual approach ensures that search systems can effectively bridge the gap between user intent and relevant content across a wide range of query types and complexities.

1.2.3.1 Sparse Retrieval

Sparse retrieval models form the backbone of classical IR systems. They represent documents and queries as high-dimensional sparse vectors, where most dimensions are zero, and focus on exact token matching rather than semantics. The term "sparse" refers to the characteristic that most positions in these high-dimensional vectors contain zero values, corresponding to terms that do not appear in a given document or query. This sparsity enables both computational efficiency and storage optimization, making sparse retrieval particularly suitable for large-scale systems where resource efficiency is paramount.

We introduced basic concepts of keyword search (Boolean retrieval), TF-IDF, and BM25 in Section 1.1.2.2. This section provides comprehensive details of these three foundational sparse retrieval methods, demonstrating their evolution from simple exact matching to sophisticated statistical ranking algorithms. To illustrate the progression and differences between sparse retrieval methods, we'll use a consistent example throughout the section.

The setup of the query and document collections for the retrieval problem setting is as follows: The query is: "machine learning optimization" and the total number of documents is: 10,000 computer science research papers.

Table 1.3 Comparison of sparse retrieval vs. dense retrieval.

Feature	Sparse retrieval	Dense retrieval
Representation	Sparse (term frequency)	Dense (neural embeddings)
Matching method	Lexical (token overlap)	Semantic (vector similarity)
Index type	Inverted Index	Vector index (e.g., FAISS)
Vocabulary mismatch	Problematic	Robust to synonyms and paraphrases
Scalability	Highly efficient	Requires vector indexing and ANN
Deep context handling	None	Enabled via transformers
Query–doc interaction	None (inverted lists)	Implicit (two-tower) or explicit (cross-attention)

The document collection includes papers on various Computer Science topics, with a sample of documents:

```
Document 1: "Machine Learning Optimization in Deep Networks" (200 words)
Document 2: "Advanced Machine Learning Techniques" (120 words)
Document 3: "Database Systems and Query Optimization" (180 words)
Document 4: "Computer Vision Applications" (150 words)
Document 5: "Optimization Algorithms for Machine Learning" (220 words)
```

1.2.3.1.1 Basic Keyword Search

Keyword search represents the simplest form of IR, operating on the principle of lexical matching. Early search systems counted term occurrences, ranking documents by the number of query words they contained.

The naive form of keyword-based search is Boolean retrieval, which evaluates queries using logical operators such as AND, OR, and NOT. It strictly matches query terms against documents without considering TF or semantic meaning.

For our query "ML optimization," we can construct different Boolean expressions as listed in Table 1.4.

Let's examine which documents match each Boolean query variation in Table 1.5:

Table 1.4 Query variations for "ML optimization."

Query variation	Boolean expression	Requirements	Approach
Variation 1	`"machine" AND "learning" AND "optimization"`	Requires ALL three terms to be present	Most restrictive approach
Variation 2	`("machine" AND "learning") OR "optimization"`	Documents must contain either both "machine" and "learning" OR contain "optimization"	More flexible approach
Variation 3	`"machine" OR "learning" OR "optimization"`	Documents need only contain ANY of the three terms	Most permissive approach

Table 1.5 Document query results for search terms.

Document	Contains "machine"	Contains "learning"	Contains "optimization"	Query_1 (AND)	Query_2 (OR)	Query_3 (OR)
Doc 1	✓	✓	✓	✓	✓	✓
Doc 2	✓	✓	×	×	✓	✓
Doc 3	×	×	✓	×	✓	✓
Doc 4	×	×	×	×	×	×
Doc 5	✓	✓	✓	✓	✓	✓

So the Boolean retrieval results are:

- Query 1 (AND): Documents 1, 5 (most restrictive)
- Query 2 (Mixed): Documents 1, 2, 3, 5 (balanced)
- Query 3 (OR): Documents 1, 2, 3, 5 (most permissive)

1.2.3.1.2 TF-IDF

To improve upon binary logic, as introduced in Section 1.1.2.2, TF-IDF provides a numerical weight to each term, balancing how important a word is to a specific document versus the entire corpus. This approach addresses the fundamental question: how important is a term for characterizing a document within a collection?

The definition of TF-IDF is as follows:

- TF measures how often a term appears in a document. It gives more weight to terms that are frequent in the document, assuming they are more important.

$$TF(t, f) = \frac{\text{Number of occurrences of the term "t" in the document "d"}}{\text{Total number of terms in the document "d"}}$$

- IDF measures the rarity of a term across the entire document collection. It reduces the weight of terms that occur in many documents, highlighting terms that are more unique to specific documents.

$$IDF(t, D) = \frac{\text{Total number of documents in the collection "D"}}{\text{Number of documents containing the term "t"}}$$

- TF-IDF combines TF and IDF to provide a balanced measure of term importance within a document and across the collection. Terms with high TF-IDF values are both frequent in a specific document and rare in the overall collection, making them highly relevant for distinguishing that document in the context of a query.

$$\text{TF} - \text{IDF(t, d, D)} = \text{TF(t, d)} \times \text{IDF(t, D)}$$

For the collection of documents listed in the example, here are the statistics for the query terms:

- "machine": appears in 3,000 of 10,000 documents
- "learning": appears in 800 of 10,000 documents
- "optimization": appears in 150 of 10,000 documents

So the IDF values for these three terms are computed as follows listed in Table 1.6:

Table 1.6 IDF computation for query terms.

Term	Docs containing term	IDF (log(10,000/df)) computation	IDF score
"machine"	3,000	log(10,000/3,000)	0.52
"learning"	800	log(10,000/800)	1.10
"optimization"	150	log(10,000/150)	1.82

Table 1.7 TF-IDF computation for three documents.

Term	IDF	Document 1 (200 words)	Document 2 (120 words)	Document 5 (220 words)
		Count/TF/TF-IDF	Count/TF/TF-IDF	Count/TF/TF-IDF
"machine"	0.52	8/0.040/0.021	4/0.033/0.017	6/0.027/0.014
"learning"	1.1	12/0.060/0.066	8/0.067/0.074	10/0.045/0.050
"optimization"	1.82	15/0.075/0.137	0/0.000/0.000	18/0.082/0.149
Total TF-IDF Score		0.224	0.091	0.213

The TF-IDF retrieval results are listed in Table 1.7. Here are the ranked results by the total TF-IDF score:

```
Document 1: 0.224 - "Machine Learning Optimization in Deep Networks"
Document 5: 0.213 - "Optimization Algorithms for Machine Learning"
Document 2: 0.091 - "Advanced Machine Learning Techniques"
Other documents: < 0.05 (not containing all query terms)
```

This example demonstrates how TF-IDF naturally identifies terms that are both locally important and globally distinctive, making them valuable for retrieval and ranking.

1.2.3.1.3 BM25

Based on TF-IDF, BM25 introduces tunable parameters that prevent TF from growing unboundedly and account for document length normalization. This prevents artificially long documents from dominating search results simply due to term repetition.

In the mathematical formulation of BM25, parameters k_1 and b control TF saturation and document length normalization, respectively:

$$\text{BM25}(q, d) = \sum \text{IDF}(q_i) \times \frac{f(q_i, d) \times (k_1 + 1)}{f(q_i, d) + k_1 \times \left(1 - b + b \times \frac{|d|}{\text{avgdl}}\right)}$$

where:

- $f(q_i, d)$ is the frequency of query term q_i in document d
- $|d|$ is the length of document d
- avgdl is the average document length in collection
- k_1 is the term frequency saturation parameter (typically 1.2–2.0)
- b is the document length normalization parameter (typically 0.75)

In the example listed, we use the below parameters setting:

```
k_1 = 1.5 (term frequency saturation)
b = 0.75 (document length normalization)
avgdl = 180 (average document length in collection)
```

Table 1.8 lists the BM25 computation results for three documents.

Table 1.8 BM25 computation for three documents.

	Document 1 (200 words)	**Document 2 (220 words)**	**Document 5 (120 words)**
Length normalization	1.11	0.75	1.17
Term	f(t,d)/TF Comp/BM25	f(t,d)/TF Comp/BM25	f(t,d)/TF Comp/BM25
"machine"	8/1.89/0.98	4/1.54/0.80	6/1.73/0.90
"learning"	12/2.14/2.35	8/2.11/2.32	10/2.12/2.33
"optimization"	15/2.25/4.10	0/0.00/0.00	18/2.29/4.17
Total BM25 score	7.43	3.12	7.4

Table 1.9 Comparison of Boolean retrieval, TF-IDF and BM25.

Method	**Boolean**	**TF-IDF**	**BM25**
Matching type	Exact	Statistical	Probabilistic
Ranking	No	Yes	Yes
Length norm	No	No	Yes
TF saturation	No	Linear	Nonlinear
Best for	Precise filtering	General search	Web-scale search

The retrieved results ranked by the BM25 scores are:

```
Document 1: 7.43 - "Machine Learning Optimization in Deep Networks"
Document 5: 7.40 - "Optimization Algorithms for Machine Learning"
Document 2: 3.12 - "Advanced Machine Learning Techniques"
Other documents: < 2.0 (lower relevance scores)
```

For the above three sparse methods, Table 1.9 lists their comparisons. While the sparse retrieval approach works well for straightforward factual queries, it struggles with semantic nuances and synonymous expressions. The limitations become apparent when users search for "car maintenance," but relevant documents discuss "automobile servicing" or "vehicle upkeep." Basic keyword search fails to recognize these semantic relationships, often missing highly relevant content due to vocabulary mismatch.

1.2.3.2 Dense Retrieval

Dense retrieval represents a fundamental shift from traditional lexical matching to semantic understanding through continuous vector representations. Unlike sparse methods that rely on exact-term matching, dense retrieval models encode both queries and documents into fixed-dimensional dense vectors, which capture semantic relationships through neural network architectures.

1.2.3.2.1 Embeddings and Granularities

As one of the major methods for dense vector representations, **embeddings** are continuous, fixed-size vector representations of linguistic units (words, phrases, sentences, or documents). They map semantically similar items to nearby points in a high-dimensional space, enabling models to reason about meaning numerically.

Word embeddings are vector representations of words that capture syntactic and semantic information based on their context in large corpora. Each word is mapped to a continuous vector in a high-dimensional space (e.g., more than 100 dimensions), where semantically similar words lie close together.

For example, we can obtain the word embeddings for words "king," "queen," "man," and "woman." As shown in the heatmap of Figure 1.16, each word can be represented by a 10-dimensional numerical vector. Using embeddings allows the model to understand similarities and relationships between different words based on their proximity in this vector space. Embeddings support vector arithmetic for analogies. For instance, as shown in Figure 1.17, in a well-trained embedding space, "king" minus "man" plus "woman" might result in a vector close to "queen," with a cosine similarity value of the first five dimensions being equal to 0.8561.

As listed in Table 1.10, embeddings are of multiple levels of granularity, capturing semantic information at varying scopes:

- Word Embeddings: Encode individual words, reflecting their semantic meanings and relationships. These embeddings capture the basic semantic units that can be combined to understand larger contexts.
- Sentence Embeddings: Capture contextual nuances of whole sentences. They aggregate word-level information to represent the meaning of sentences, considering word order, and syntactic structure.
- Document Embeddings: Represent entire documents, summarizing overall themes, and information. Document embeddings provide a holistic view of the content, making them useful

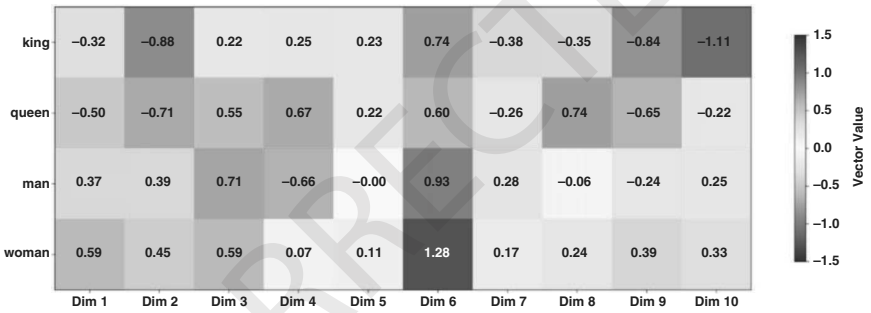


Figure 1.16 Word embeddings (first 10 dimensions) for four words: king, queen, man, and woman.

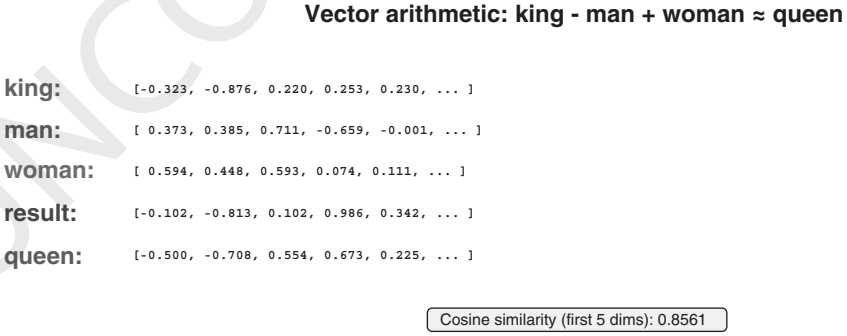


Figure 1.17 Vector arithmetic and similarity metrics.

Table 1.10 Different levels of embedding granularities.

Level	Description
Word embeddings	Represent individual words (e.g., "king," "queen"). Captures basic semantics
Sentence embeddings	Encodes full sentences. Useful for tasks like paraphrase detection or retrieval
Document embeddings	Aggregates multiple sentences. Suitable for long-form retrieval tasks

for retrieving documents that match the overall query intent. Sentence embeddings combine word-level semantics into a unified representation, while document embeddings aggregate sentence-level embeddings to capture global meaning.

Some popular pretrained embeddings include the following:

- OpenAI's Text-embedding-3-small/large: General-purpose, high-accuracy, LLM-aligned vectors.
- BAAI's BGE Models: Multilingual, sentence-level embeddings that support retrieval and reranking across diverse languages and topics.

We are going to discuss more details of how to obtain embeddings in Section 2.1.2.2 of this book.

1.2.3.2.2 Embedding-based Retrieval

Embedding-based retrieval (EBR) is a retrieval paradigm where documents and queries are represented as dense vectors (embeddings) in a latent space. Retrieval is then performed by comparing these vector representations using a similarity function, such as cosine similarity or inner product. This approach allows the system to capture semantic similarity between queries and documents beyond exact keyword matching, enabling more flexible and meaningful retrieval. Modern EBR systems rely on neural networks to generate the embeddings and often use specialized vector indexes to efficiently find nearest-neighbor vectors corresponding to relevant documents.

By encoding queries and documents into dense vectors, EBR facilitates relevance ranking based on semantic similarity rather than exact term matches. This section explores the key architectures, methodologies, and trade-offs in EBR.

Basic Vector Index Retrieval is the foundational approach in EBR, where documents and queries are encoded into dense embeddings using neural networks, and retrieval is performed by searching for the nearest document embeddings to a given query embedding in the vector space. This process typically involves the following steps:

1. Embedding Generation:
   - Query Embedding: Let $q \in \mathbb{R}^d$ be the embedding of the query, computed as $q = f_q(\text{query})$, where f_q is a neural network (e.g., BERT).
   - Document Embedding: Let $d_i \in \mathbb{R}^d$ be the embedding of document D_i, computed as $d_i = f_d(D_i)$, where f_d is a similar or identical neural network.
2. Similarity Function: Cosine similarity between query and document embeddings:

$$\text{sim}(q, d_i) = \frac{q \cdot d_i}{\| q \| \cdot \| d_i \|}$$

3. Retrieval: Return the top-k documents with the highest similarity scores. We can use an index (e.g., FAISS) to approximate the nearest neighbors (ANN) efficiently.

$$\text{Top-}k = \arg\max_{i=1,\ldots,n} \text{sim}(q, d_i)$$

For efficiency considerations, a straightforward dense vector search can be accelerated by vector indexes like inverted file indexes (IVF) or product quantization [9], which cluster or quantize embeddings to limit comparisons. Another approach is using graph-based structures. For example, hierarchical navigable small world (HNSW) [10] has emerged as the de facto standard for high-dimensional approximate nearest neighbor search, combining navigable small world graphs with hierarchical structures inspired by skip lists. These methods dramatically reduce search latency compared to exhaustive comparison, making real-time retrieval feasible even for millions of documents. However, basic vector indexes may still struggle as data scales, which motivates more structured or hierarchical retrieval strategies.

1.2.3.2.3 Two-tower Architecture

The two-tower model is the most widely adopted neural architecture for EBR due to its balance of efficiency, scalability, and performance. It consists of two independent neural networks, or "towers," that encode queries and documents separately into a shared embedding space. The architecture is designed to optimize both real-time query processing and offline document indexing, making it ideal for large-scale retrieval systems.

As illustrated in Figure 1.18, the Two-tower Architecture comprises:

1. Query Tower:
   - Purpose: Encodes user queries into dense embeddings that capture user intent and context.
   - Design Considerations: (1) query understanding: incorporates techniques like query expansion (e.g., synonym augmentation) and contextual analysis to enhance semantic understanding. (2) Low latency: optimized for real-time processing to ensure fast response times in interactive applications. (3) Input features: typically processes raw query text, metadata (e.g., user context), or reformulated queries.
2. Document Tower:
   - Purpose: Encodes documents into embeddings that represent their content and semantics comprehensively.
   - Design Considerations: (1) content understanding: Captures the full context of the document, including text, structure, and metadata (e.g., titles, tags). (2) Pre-computation: designed for offline processing, allowing embeddings to be computed and indexed in advance. (3) Scalability: handles large document collections by leveraging batch processing and distributed computing.

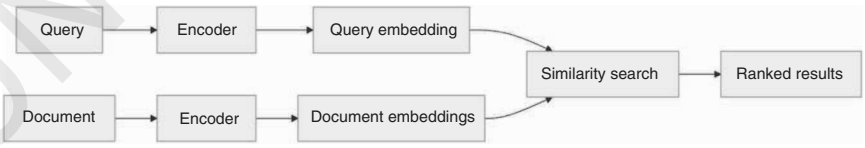


Figure 1.18 Two-tower architecture.

The two-tower model is typically trained using contrastive learning, where the objective is to maximize the similarity between embeddings of relevant query-document pairs while minimizing the similarity for irrelevant pairs. Common loss functions include:

- Contrastive Loss: Encourages positive pairs (relevant query-document) to be closer in the embedding space than negative pairs (irrelevant query-document).
- Triplet Loss: Uses triplets of (query, relevant document, irrelevant document) to enforce ranking.
- In-batch Negatives: Leverages other documents in the same batch as negative examples to improve training efficiency.

1.2.3.2.4 Cross-attention Models

While two-tower models prioritize efficiency, cross-attention models focus on maximizing relevance by enabling deep interactions between query and document representations. These models process query-document pairs jointly, allowing the model to capture fine-grained relationships at the token level.

As shown in Figure 1.19, the cross-attention architecture follows this structure: Query + Document → Joint Encoder → Relevance Score

- Joint Encoder: A single neural network (typically a transformer) processes the concatenated query and document text, using cross-attention mechanisms to model interactions between query tokens and document tokens.
- Cross-attention Mechanism: Each token in the query attends to all tokens in the document (and vice versa), allowing the model to capture context-specific relationships.
- Output: A relevance score (e.g., a scalar value) indicating the likelihood that the document is relevant to the query.

The choice and implementation of retrieval models significantly impact system performance, both in terms of relevance and efficiency. IR systems often employ multiple retrieval approaches, carefully balanced to achieve optimal results across different query types and use cases.

1.2.3.2.5 Hybrid Methods

The choice and implementation of retrieval models have a profound impact on system performance, influencing both relevance and efficiency. Table 1.11 lists the comparisons of the two-tower architecture vs. the cross-attention architecture.

IR systems therefore often employ multiple retrieval approaches in a coordinated pipeline to achieve optimal results across different scenarios. A common pattern is a two-stage (or multistage) retrieval process:

- Stage 1—High-recall Retrieval: Use a fast but less expensive method (like a two-tower EBR model with an ANN index, or even a traditional keyword search) to retrieve a broad set of candidates. This stage prioritizes recall—getting in as many potentially relevant items as possible with minimal latency. The two-tower model is ideal here because it can handle

Figure 1.19 Cross-attention architecture.

Table 1.11 Trade-off of two-tower vs. cross-attention models.

Factor	Two-tower	Cross-attention
Latency	1–10 ms	100–1000 ms
Accuracy	Good	State of the art
Scalability	Billions of docs	Thousands of docs
Freshness	Instant	Retraining needed

massive candidate sets quickly, outputting embeddings that are matched via ANN search in milliseconds.
- Stage 2—Precise Reranking: Take the candidate set from Stage 1 and apply a more powerful but slower model to refine the ordering. This could be a cross-attention neural reranker or another model that considers richer features and interactions. This stage focuses on precision—ensuring the very top results are truly the best answers for the query. Because this stage deals with far fewer items (hundreds or less instead of millions), it can afford the computational cost of deep models.

This hybrid approach balances efficiency and accuracy, making it suitable for diverse query types and large-scale deployments.

1.2.4 Post-retrieval

While retrieval models efficiently identify potentially relevant documents, the post-retrieval phase ensures that results are not only technically relevant but also contextually meaningful and personalized to user intent. The post-retrieval stage transforms these raw candidates into a refined, ranked, and user-aligned output. This phase comprises three critical components:

- **Ranking**, which decides the order of the results based on predicted relevance.
- **Result representation**, which determines how results are communicated to users.
- **Evaluation**, which ensures the system's performance meets expectations via metrics and feedback.

Through iterative refinement driven by user feedback, post-retrieval processes enable IR systems to adapt dynamically, ensuring precision and relevance in diverse contexts.

1.2.4.1 Ranking

While retrieval models identify potentially relevant documents, ranking systems determine the order in which they are presented to the user. This process is crucial for ensuring that the most relevant documents appear at the top of the search results. The goal is to produce a preliminary ranked list that captures broadly relevant documents while maintaining scalability.

1.2.4.1.1 Learning-to-rank

Learning-to-rank (L2R) is a family of ML techniques that optimize rankings using labeled datasets [11]. There are several approaches within L2R that have significantly influenced ML-based search methods:

- Pointwise Approaches: These convert ranking problems into regression or classification tasks on individual documents. By treating each document independently, pointwise approaches

aim to assign a direct relevance score based on the document's content and the query. For example, a linear regression model predicts a relevance score with this loss function:

$$\mathcal{L}_{\text{pointwise}} = \sum_i (s_i - y_i)^2$$

where s_i is the predicted score, and y_i is the true relevance.

- Pairwise Approaches: These treat ranking as a classification problem between pairs of documents. This method evaluates which of two documents is more relevant to a specific query, refining the accuracy of rankings by learning from these comparisons. For example, RankSVM [12] minimizes a hinge loss:

$$\mathcal{L}_{\text{pairwise}} = \sum_{(i,j)\in\text{pairs}} \max(0, 1 - (s_i - s_j))$$

  RankNet [13] minimizes a cross-entropy loss:

$$\mathcal{L}_{\text{pairwise}} = -\sum_{(i,j)\in\text{pairs}} \log \sigma(s_i - s_j)$$

  Other notable pairwise ranking methods include LambdaRank [14].
- Listwise Approaches: These consider the entire list of documents directly. By modeling the ranking process in a holistic manner, listwise approaches aim to overcome the limitations of pointwise and pairwise methods, offering a more nuanced and effective way to manage rankings in search results. For instance, ListNet [15] minimizes a loss based on the permutation probability of the ranked list:

$$\mathcal{L}_{\text{listwise}} = -\sum_{\pi} P(\pi|y) \log P(\pi|s)$$

  where π is a permutation, $P(\pi|y)$ is the ground-truth permutation probability, and $P(\pi|s)$ is the predicted probability. Other notable listwise ranking methods include ListMLE [16] and LambdaMART [17].

Overall, the Pointwise method is efficient but ignores the relative relationship between documents; Pairwise improves local accuracy by comparing document pairs, while Listwise optimizes globally but has a higher computational cost.

Modern IR systems commonly employ a two-stage ranking architecture, consisting of an initial ranking phase followed by a reranking phase. Each stage plays a distinct and complementary role in balancing efficiency and accuracy in large-scale systems.

1.2.4.1.2 Initial Ranking: Fast, Scalable Candidate Selection

The initial ranking stage is responsible for rapidly generating a shortlist of candidate documents or items from a potentially massive index—often millions or billions of entries. The focus here is on speed and recall, ensuring that the set of retrieved documents contains as many relevant candidates as possible.

While fast and scalable, initial ranking models typically rely on limited feature sets, avoid complex computations, and may not fully model fine-grained user intent or context. This can lead to relevant results being improperly ranked lower or less relevant results appearing near the top.

1.2.4.1.3 Re-ranking: Precision-oriented Refinement

The reranking stage refines the initial ranked list by reassessing documents using additional signals and more computationally intensive models. Reranking is essential because it addresses

the limitations of initial ranking by incorporating contextual, user-specific, and temporal factors that enhance precision and relevance. By focusing on a smaller subset of top-ranked documents (e.g., top 100), reranking balances computational cost with improved ranking quality, making it a critical step in modern IR systems.

Multifactor reranking combines relevance scores with additional signals through learned weighting functions. A typical implementation might use:

$$\text{Final\_Score} = \alpha \times \text{Relevance\_Score} + \beta \times \text{Freshness\_Score} + \gamma \times \text{Authority\_Score} + \delta \times \text{Personalization\_Score}$$

where the weights $\alpha, \beta, \gamma, \delta$ are learned from user interaction data (e.g., clicks, dwell time, conversions) and system performance metrics (e.g., NDCG, CTR). These weights allow the ranking function to dynamically prioritize different aspects of document quality, such as content relevance, recency, trustworthiness, and user-specific preferences.

By combining initial ranking and reranking, systems ensure that retrieved documents are not only relevant but also optimally ordered to meet user expectations. However, the effectiveness of ranking depends on how results are presented to users, which we explore in Section 1.2.4.2 on result representation.

1.2.4.2 Result Representation

Once documents are ranked, the result representation stage determines how they are presented to the user, significantly influencing the overall experience. Effective result representation goes beyond listing documents; it involves structuring and visualizing results to enhance usability, interpretability, and engagement. This stage is critical for minimizing the user's cognitive load and helping them quickly assess relevance without clicking on every link.

Key components of result representation include:

1.2.4.2.1 Snippets and Highlighting

Instead of just showing a title, the system generates a short snippet—a few lines of text from the document that contains the user's query terms. To draw the eye, these query terms are often bolded, providing an immediate visual cue of the document's relevance.

1.2.4.2.2 Rich Results and Knowledge Panels

For certain types of queries, the system can display structured data in a more visually appealing and informative format.

1.2.4.2.3 Personalized Formatting

Recommendation systems tailor result layouts based on user preferences. For example, a streaming service might prioritize movie posters for visual learners or detailed synopses for text-focused users.

1.2.4.2.4 Contextual Highlighting

Highlighting query terms or related concepts in results enhances relevance perception. For example, a news aggregator might bold keywords like "climate change" in article summaries.

For example: In a question-answering system, a query like "What is transformer architecture?" might present results as:

- A concise answer extracted from the top-ranked document: "The transformer is a neural network architecture using self-attention..."

- A list of related articles with snippets and links.
- A visual diagram (e.g., a transformer model schematic) to aid understanding.

In summary, effective representation turns a list of documents into a rich, interactive, and efficient user experience, bridging the final gap between the system's output and the user's understanding.

1.2.4.3 Evaluation

After the ranking and presentation phases, the system produces its final output—whether a ranked list of documents, recommended items, or responses in an interactive system. To ensure high-quality results and continuous improvement, a robust evaluation framework is essential. Evaluation serves as the critical feedback mechanism that measures how well the system meets user needs and achieves its intended goals.

The evaluation process is not a one-time activity but an ongoing cycle that feeds back into the system. By analyzing evaluation results, the system can identify areas for improvement and adapt its retrieval and ranking mechanisms accordingly. This feedback loop is a key feature of modern IR systems, enabling them to learn from user interactions and evolve over time.

1.2.4.3.1 Evaluation Metrics

Evaluation metrics fall into two major categories: offline metrics and online metrics, each tailored to specific use cases. Offline metrics are computed using pre-labeled datasets; online metrics capture real-time user feedback via A/B testing or behavioral signals.

1. **Offline Evaluation**: Offline evaluation measures how well a system performs on known test data with labeled relevance. Common metrics include:
   - **Precision**: The proportion of retrieved documents that are relevant:

     $$\text{Precision} = \frac{|\text{Relevant Documents} \cap \text{Retrieved Documents}|}{|\text{Retrieved Documents}|}$$

     Example: For a query retrieving 10 documents, 7 of which are relevant, precision is $\frac{7}{10} = 0.7$.
   - **Recall**: The proportion of relevant documents retrieved:

     $$\text{Recall} = \frac{|\text{Relevant Documents} \cap \text{Retrieved Documents}|}{|\text{Relevant Documents}|}$$

     Example: If there are 12 relevant documents in the collection and 7 are retrieved, recall is $\frac{7}{12} \approx 0.583$.
   - **Normalized Discounted Cumulative Gain (NDCG)**: Evaluates ranking quality by weighting relevance by position:

     $$\text{DCG}_k = \sum_{i=1}^{k} \frac{\text{rel}_i}{\log_2(i+1)}, \quad \text{NDCG}_k = \frac{\text{DCG}_k}{\text{IDCG}_k}$$

     where rel_i is the relevance score of the document at position i, and IDCG_k is the ideal DCG for the top-k documents. Example: For a ranked list with relevance scores [5, 13, 18], DCG is computed and normalized against the ideal ranking [5, 13, 18].

- **Mean Average Precision (MAP)**: Averages precision across all relevant documents for multiple queries:

$$\text{AP} = \frac{\sum_{k=1}^{n} (\text{Precision@}k \cdot \text{rel}_k)}{|\text{Relevant Documents}|}, \quad \text{MAP} = \frac{\sum_{q \in Q} \text{AP}_q}{|Q|}$$

  Example: For a query with three relevant documents retrieved at ranks 1, 3, and 5, AP considers precision at each relevant position.
- **Reciprocal Rank (RR)**: The inverse of the rank of the first relevant document:

$$\text{RR} = \frac{1}{\text{Rank of First Relevant Document}}$$

  Example: If the first relevant document is at rank 2, RR is $\frac{1}{2} = 0.5$.

2. **Online Evaluation:** Live systems require user-centered metrics that capture behavior and engagement.
   - **Click-through Rate (CTR)**: The proportion of users who click on a result:

$$\text{CTR} = \frac{\text{Number of Clicks}}{\text{Number of Impressions}}$$

     Example: If 100 users see a recommended movie and 20 click, CTR is $\frac{20}{100} = 0.2$.
   - **Dwell Time**: The average time users spend engaging with a result. For example: In a news aggregator, users spending 30 seconds on an article indicate higher engagement than those spending five seconds.
   - **Session Duration**: Measures the total time spent interacting with the system.
   - **Conversion Rate**: The proportion of users who take a desired action (e.g., purchasing a product):

$$\text{Conversion Rate} = \frac{\text{Number of Conversions}}{\text{Number of Users}}$$

     Example: In an e-commerce platform, if 50 out of 1000 users buy recommended headphones, the conversion rate is $\frac{50}{1000} = 0.05$.

Modern evaluation approaches also incorporate A/B testing to compare system variants with real users, and online evaluation methods that capture real-time user feedback and behavioral signals.

User feedback collected during evaluation dynamically adjusts ranking models, ensuring continuous improvement in alignment with user intent. The evaluation framework described here forms the basis for the advanced Graph-RAG system evaluations detailed in Chapter 9.

1.3 Information Retrieval Applications

IR powers a diverse ecosystem of modern applications, from everyday web searches to personalized content recommendations. While these applications share fundamental technical components such as document indexing, candidate retrieval, and result ranking (as discussed in Section 1.2.1), they differ dramatically in their approach to understanding and satisfying user needs.

The most prominent IR applications fall into two primary categories: **search systems** and **recommendation systems**. These represent fundamentally different paradigms for connecting users with relevant information. Search systems respond to explicit user queries where individuals clearly articulate their information needs through typed or spoken input. In contrast, recommendation systems operate without explicit queries, instead inferring user preferences from behavioral patterns, contextual signals, and historical interactions to proactively suggest relevant content.

Table 1.12 Core differences of search and recommendation systems.

Aspect	Search systems	Recommendation systems
User input	Explicit queries (e.g., "best hiking boots")	Implicit signals (past behavior, context)
Primary goal	Match documents to query intent	Anticipate user preferences
Key metrics	Precision, NDCG	Engagement, retention
User intent	Immediate information need	Discovery of new interests
Example	Google search	Netflix "Top Picks for You"

This fundamental distinction in user input processing cascades through every aspect of system design, from the algorithms used for understanding user intent to the metrics employed for evaluating success. Understanding these differences is crucial for designing effective IR systems and selecting appropriate techniques for specific application domains. As shown in Table 1.12, each paradigm serves distinct user needs and employs different approaches to IR.

Beyond these primary applications, IR techniques extend to specialized domains, including targeted advertising systems, AI-powered assistants (such as Siri and Alexa), and domain-specific platforms serving healthcare, legal research, and e-commerce contexts. Each of these applications adapts core IR principles to meet specific user needs and operational constraints.

This section explores search and recommendation systems in detail, examining their distinct approaches to user input processing, core architectural components, and unique technical challenges. Through practical examples and evaluation strategies, we demonstrate how these systems translate theoretical IR concepts into real-world solutions that serve billions of users daily.

1.3.1 Search Systems

Search systems represent the most direct application of IR principles, designed to retrieve and rank documents in response to explicit user queries. The availability of clear textual input makes query understanding and relevance matching central to their operation, creating a pipeline that must effectively transform user intentions into relevant document selections.

The explicit nature of search queries provides both opportunities and challenges. While users clearly express their information needs, the system must accurately interpret potentially ambiguous language, handle variations in query formulation, and understand implicit context that users may not explicitly state. This requirement drives sophisticated query processing techniques that form the foundation of modern search systems.

1.3.1.1 Search System Architecture

The search pipeline consists of four interconnected stages, each specifically tailored to handle explicit user input effectively and ensure that results align closely with the query's intent:

1. **Query Processing**: Transforms raw user input into structured representations suitable for retrieval.
2. **Retrieval**: Identifies candidate documents from large collections using both sparse and dense methods.
3. **Ranking**: Scores and orders candidates based on relevance and quality signals.
4. **Result Presentation**: Formats and displays results to optimize user comprehension and task completion.

This architecture reflects the linear nature of search interactions, where each user query triggers a complete processing cycle designed to satisfy an immediate information need.

1.3.1.2 Query Processing: The Foundation of Search Understanding

Query processing serves as the cornerstone of search systems, transforming raw user input into structured forms that enable effective retrieval and ranking. This complex process addresses the inherent challenges of natural language input while preserving user intent throughout multiple transformation stages.

Modern query processing encompasses three interconnected areas that work together to bridge the gap between human expression and machine understanding: intent recognition, query rewriting, and routing with decomposition. Each area addresses specific aspects of the interpretation challenge while contributing to overall system effectiveness.

1.3.1.2.1 Intent Recognition: Understanding User Goals

The first critical step in query processing involves understanding the user's underlying goal, which drives all subsequent processing decisions. Search intent typically falls into three well-established categories, each requiring different retrieval strategies and result presentation formats [18]:

Informational queries seek knowledge or factual information about specific topics. When users ask, "What is the capital of France?" they expect direct, authoritative answers presented clearly and concisely. These queries benefit from integration with knowledge bases, featured snippets, and structured data that can provide immediate answers.

Navigational queries aim to reach specific websites or digital destinations. A search for "Wikipedia" indicates the user wants to access that particular platform rather than learn about wikis in general. These queries require understanding brand names, website entities, and user navigation patterns to surface the intended destination prominently.

Transactional queries express intent to complete specific actions such as purchasing products or downloading files. "Buy plane tickets to New York" signals commercial intent that benefits from integration with e-commerce platforms, pricing information, and actionable interfaces.

Modern search systems employ ML classifiers trained on large query datasets to automatically categorize intent. For example, when processing "capital of France," the system uses NLP to classify the query as informational and prioritizes factual responses from authoritative knowledge sources like Wikidata or educational institutions.

1.3.1.2.2 Query Rewriting: Enhancing Input Quality

Once intent is established, query rewriting techniques refine the user's input to improve retrieval accuracy and address common issues in natural language expression. This process involves several complementary approaches that standardize, expand, and enrich the original query.

Query normalization addresses basic input quality issues through spelling correction and term standardization. Advanced systems use techniques like Levenshtein distance calculations combined with domain-specific dictionaries to correct errors. For instance, "Azuer Devpos tutorials" becomes "Azure DevOps tutorials" through character-level corrections informed by technology terminology patterns [19].

Query compression removes redundant or irrelevant terms, particularly valuable in conversational search contexts where users provide extensive context. A verbose input like "I'm looking for something affordable. What are the best hiking trails near me?" can be compressed to "best hiking trails near me," focusing processing power on the core information need while preserving essential context.

Query expansion addresses vocabulary mismatch by adding synonyms and related terms that capture broader intent. Statistical approaches analyze query logs to identify terms that frequently co-occur in successful searches, while semantic approaches use word embeddings to find conceptually related terms. For example, "best hiking trails" might expand to include "top-rated hiking trails," "scenic walking paths," or "outdoor recreation areas" based on successful historical searches and semantic relationships [20].

Context enrichment enhances queries with additional information derived from user history, session behavior, and external knowledge sources. This process considers previous queries and clicks within the current session, long-term user preferences when available, and relevant structured data from ontologies or knowledge graphs. For instance, the original query "best laptops 2024" might be enriched to "top-reviewed laptops for students with long battery life in 2024" when the system detects educational context from the user's search history.

1.3.1.2.3 Routing and Decomposition: Managing Complex Information Needs

The final aspect of query processing involves directing queries to appropriate resources and breaking down complex information needs into manageable components.

Query routing ensures that search requests reach the most suitable data sources and processing systems based on query characteristics and system architecture. In enterprise environments, queries about customer transactions might route to secure financial databases with appropriate access controls, while general knowledge queries access public web indexes optimized for broad coverage. Effective routing considers data sensitivity, computational requirements, geographic relevance, and regulatory compliance to optimize both performance and security.

Query decomposition addresses complex queries that contain multiple information needs by breaking them into simpler sub-questions that can be processed independently. For example, "Who won the championship recently, Red Sox or Patriots?" can be decomposed into "When did the Red Sox last win a championship?" and "When did the Patriots last win a championship?" This decomposition enables parallel processing of sub-queries and more precise retrieval for each component, with answers later synthesized into a coherent response [21].

These traditional query processing techniques rely on combinations of predefined rules, statistical models, and heuristic-based enrichment methods to infer user intent and prepare queries for effective retrieval. The sophistication of this processing directly impacts the quality of subsequent retrieval and ranking stages.

1.3.1.3 Search System Challenges

Despite decades of advancement, search systems continue to face significant challenges that drive ongoing research and development efforts.

Intent ambiguity remains a persistent challenge when queries like "apple" could refer to fruit, technology companies, or record labels. Robust intent recognition requires considering user context, search history, and real-time signals to disambiguate effectively.

Scalability demands processing millions of documents and queries in real time while maintaining sub-second response times. This requirement drives innovations in efficient retrieval techniques (such as HNSW indexes discussed in Section 1.2.3.2) and distributed ranking systems.

Evaluation complexity involves measuring not just relevance accuracy through metrics like NDCG and MAP (Section 1.2.4.3), but also user satisfaction, task completion rates, and long-term engagement. Capturing nuanced intent and user context in evaluation frameworks requires ongoing refinement and adaptation to changing user behaviors.

1.3.2 Recommendation Systems

Recommendation systems represent a fundamentally different approach to IR, operating without explicit user queries and instead interpreting implicit behavioral signals to suggest relevant content proactively. This paradigm shift from reactive search to proactive discovery creates unique opportunities for content exploration while introducing distinct technical challenges.

Unlike search systems that respond to immediate, articulated information needs, recommendation systems must continuously analyze user behavior patterns, contextual signals, and content characteristics to predict what users might find valuable or engaging. This prediction challenge requires sophisticated user modeling techniques that capture both momentary interests and evolving long-term preferences.

The absence of explicit queries fundamentally changes system design priorities. While search systems optimize for query-document relevance, recommendation systems focus on user engagement, content discovery, and long-term platform retention. These different objectives drive distinct approaches to candidate generation, ranking strategies, and evaluation methodologies.

1.3.2.1 Recommendation System Architecture

The recommendation pipeline mirrors search systems in its use of retrieval and ranking components but operates from a foundation of implicit user input analysis:

1. **User Interest Modeling**: Infers preferences from behavioral signals, demographic information, and contextual factors.
2. **Candidate Retrieval**: Generates potential recommendations using collaborative filtering, content analysis, and hybrid approaches.
3. **Ranking**: Scores candidates based on predicted user preference, engagement likelihood, and strategic objectives.
4. **Result Presentation**: Displays recommendations in formats optimized for discovery and engagement.

This architecture reflects the proactive nature of recommendation systems, where user models continuously evolve and drive content suggestions without explicit user requests.

For example, TikTok's recommendation system analyzes a user's viewing history (such as frequent engagement with fitness videos) to build interest profiles that inform candidate retrieval using embedding-based techniques. The ranking stage prioritizes content with high engagement potential, while reranking ensures diversity by including content from different genres and creators.

1.3.2.2 User Interest Modeling

User interest modeling forms the foundation of effective recommendation systems, replacing the explicit query understanding required in search systems. This modeling process must infer complex, multifaceted user preferences from indirect behavioral signals and contextual information.

Temporal interest dynamics recognize that user preferences operate at multiple time scales. Short-term modeling captures immediate intent from recent interactions, such as recommending running shoes after a user views fitness gear or suggesting related cooking videos during a baking session. Long-term modeling analyzes historical data spanning weeks or months to identify stable preferences, like a consistent interest in science fiction movies or a preference for organic food products.

Multi-time scale models combine both perspectives to balance immediate relevance with personalized recommendations. Spotify, for example, might recommend a new workout playlist based on recent gym check-ins (short-term signal) while prioritizing rock music based on months of listening history (long-term preference).

Multi-interest recognition acknowledges that users typically maintain diverse, sometimes conflicting interests that cannot be captured by single preference vectors. A user might simultaneously enjoy horror movies, cooking shows, and financial news, representing distinct interest clusters that require separate modeling approaches.

Advanced systems employ techniques such as multi-head attention mechanisms and mixture-of-experts models to identify and separately model these diverse interest dimensions. This approach enables more nuanced recommendations that can appropriately switch between different user personas based on current context and interaction patterns.

1.3.2.3 Traditional Recommendation Techniques

Collaborative filtering leverages user behavior patterns to suggest items based on similarities between users or items. User-based collaborative filtering identifies users with similar interaction patterns and recommends items enjoyed by those similar users. For example, if User A and User B both rated science fiction movies highly, the system might recommend to User A other movies that User B enjoyed but User A hasn't yet seen.

Item-based collaborative filtering focuses on relationships between items rather than users, suggesting items similar to those the user has previously engaged with. This approach might recommend science fiction books based on past purchases of similar titles, using co-interaction patterns to identify related content.

While collaborative filtering excels at capturing complex preference patterns without requiring content understanding, it struggles with the cold-start problem, where new users or items lack sufficient interaction data for accurate recommendations [22]. Solutions include leveraging content-based features for new items or social signals for new users.

Content-based filtering recommends items with attributes similar to those previously engaged with by the user. This approach analyzes item features such as genre, actors, and directors for movies, or topics and authors for articles, to create detailed content profiles. User profiles are built by aggregating features from items with which they've interacted, enabling recommendations through profile matching.

While effective for users with consistent, well-defined preferences, content-based systems risk creating filter bubbles by repeatedly recommending similar content, potentially limiting exposure to diverse experiences and new interests [23].

Hybrid methods combine collaborative and content-based approaches to leverage the strengths of different methodologies while mitigating individual weaknesses. Ensemble methods blend predictions from multiple algorithms, such as averaging scores from matrix factorization and neural models. Feature augmentation uses outputs from one recommendation technique as inputs for another, such as incorporating collaborative filtering user embeddings into content-based models.

Switching strategies adapt recommendation approaches based on contextual factors, deploying popular items for new users while providing personalized suggestions for engaged users. Most production systems, including Netflix and Amazon, employ hybrid approaches to achieve robust performance across diverse user segments and content types [24].

Sequential modeling captures meaningful patterns in user behavior over time, recognizing that interactions often follow logical progressions both within individual sessions and across longer

periods. Session-based recommendations model the evolution of user interests during single interaction periods, such as suggesting the next logical video during a YouTube watching session or the next product during an e-commerce browsing session.

Sequence-aware recommenders track how preferences evolve across multiple sessions and extended time periods, such as gradually shifting music tastes from pop to indie rock or evolving reading preferences from fiction to nonfiction as life circumstances change [25].

1.3.2.4 Recommendation System Challenges

Recommendation systems face unique challenges that differ significantly from those encountered in search systems.

The cold-start problem affects both new users who lack sufficient interaction history for accurate preference modeling and new items that haven't received enough user feedback for confident recommendations. This challenge requires creative solutions such as onboarding questionnaires, social signal integration, or content-based fallback strategies.

Filter bubbles and echo chambers can result from overreliance on past behavior patterns, potentially limiting user exposure to diverse content and new experiences. Balancing personalization with diversity requires sophisticated techniques that encourage exploration while maintaining engagement.

Evaluation complexity involves measuring success through online metrics such as CTRs, dwell time, and long-term user retention (Section 1.2.4.3). However, optimizing for immediate engagement may conflict with longer-term goals of user satisfaction, content discovery, and platform diversity. This tension requires careful balancing of multiple, sometimes competing objectives.

1.3.3 Example: WANDS Dataset

To illustrate the practical application of IR techniques throughout the remainder of this book, we will utilize the WANDS (wayfair product search relevance dataset) as a concrete example that demonstrates both search and recommendation system principles in an e-commerce context.

The WANDS dataset provides an open-source benchmark specifically designed to evaluate the relevance of search results in real-world e-commerce scenarios. This dataset encompasses the complexities of product search, including diverse query types, varied product attributes, and nuanced relevance judgments that reflect actual user behavior patterns.

By grounding our theoretical discussions in this practical dataset, we can demonstrate how the concepts explored in this chapter—from query processing techniques to user interest modeling—translate into measurable improvements in system performance. The dataset's availability at https://github.com/wayfair/WANDS enables readers to explore these techniques hands-on, bridging the gap between academic understanding and practical implementation.

This practical foundation will prove particularly valuable as we explore advanced topics such as neural retrieval architectures, ranking optimization, and evaluation methodologies in subsequent chapters, providing concrete examples that illustrate both the potential and the challenges of modern IR systems.

1.3.3.1 Dataset Overview and Structure

The WANDS dataset represents a comprehensive e-commerce search benchmark that captures the real-world complexity of product discovery. Understanding its structure provides the foundation for implementing and evaluating IR techniques in commercial contexts.

1.3.3.1.1 Core Dataset Components

The WANDS dataset is organized into three primary files that together form a complete search evaluation framework:

Product catalog (`product.csv`) contains detailed information about 42,994 products, including:

Product ID: Unique identifier for each product
Title: Customer–facing product name
Class: Product category classification
Category Hierarchy: Nested taxonomy path
Description: Detailed product description with specifications
Features: Product attributes and specifications as structured key–value pairs
User Ratings: Average customer rating on a 1–5 scale
Reviews: Collection of user–generated reviews with sentiment indicators

Query collection(`query.csv`) encompasses 480 diverse search queries representing authentic user search behaviors:

Query ID: Unique identifier for each search query
Raw Query Text: Original user input without preprocessing
Associated Class: Primary product category the query targets

Relevance judgments (`label.csv`) provide 233,448 human-annotated relevance assessments using a three-level scale:

Exact: Product perfectly matches query intent and user need
Partial: Product partially satisfies query intent with some relevance
Irrelevant: Product does not match query intent or user need

1.3.3.1.2 Dataset Statistics and Characteristics

Table 1.13 lists the key statistics and characteristics of the datasets.

And here is an example product entry:

```
# Example Query Entry
"query.csv": [
    {
      "query_id": "0",
      "query": "salon chair",
      "query_class": "Massage Chairs"
    },
    {
      "query_id": "1",
      "query": "smart coffee table",
      "query_class": "Coffee & Cocktail Tables"
    }
  ],

# Example Product Entry
"product.csv": [
```

Table 1.13 WANDS dataset file structure and contents.

File	Contents	Records	Key Fields
`product.csv`	Product details	42,994	ID, title, category, description, features
`query.csv`	Customer searches	480	Query ID, text, class
`label.csv`	Relevance labels	233,448	Query ID, Product ID, relevance score

```
    {
      "product_id": 1,
      "product_name": "all-clad 7 qt. slow cooker",
      "product_class": "Slow Cookers",
      "category_hierarchy": "Kitchen & Tabletop / Small Kitchen Appliances / Pressure & Slow Cookers / Slow Cookers / Slow Slow Cookers",
      "product_description": "create delicious slow-cooked meals, from tender meat to flavorful veggies , with this easy-to-use slow cooker . the unit features a nonstick cast-aluminum insert that moves seamlessly from the oven or stovetop to the electric base to the table . you can use the insert alone or with the slow cooker to make a variety of one-pot dishes from soup to desserts , and much more . you can even prepare your ingredients in the morning , place everything in the slow cooker , and walk away to come home to the aroma of a hot , healthy dinner at the end of a busy day . with its sleek stainless-steel finish , the slow cooker not only presents beautifully , but it' s also the perfect size to accommodate the whole family or a large group when entertaining .",
      "product_features": {
        "capacity_quarts": "7",
        "product_type": "slow cooker",
        "programmable_cooking_settings": ["slow cook", "keep warm"],
        "overall_height_top_to_bottom": "14.2",
        "overall_depth_front_to_back": "11.3",
        "programmable_food_settings": ["soup", "stew"],
        "overall_width_side_to_side": "19.9",
        "top_programmable_settings": ["soup & stew"],
        "dishwasher_safe_parts": ["cooking pot/insert", "lid"],
        "features": ["non-stick", "automatic shutoff", "adjustable temperature settings"],
        "housing_heating_base_material": "stainless steel",
        "cooking_pot_insert_material": "aluminum",
        "operation_type": "programmable",
        "finish": "white",
        "overall_product_weight": "13",
        "uniform_packaging_and_labeling_regulations_uplr_compliant": "no",
        "commercial_warranty": "no",
        "indicator_light": "yes",
        "removable_cooking_pot_insert": "yes",
        "programmable": "yes",
        "country_of_origin": "china",
        "supplier_intended_and_approved_use": "residential use"
      },
      "rating_count": 100,
      "average_rating": 2.0,
      "review_count": 98
    }
]

# Example Label Entry
"label.csv": [
    {
      "id": 3001,
      "query_id": 27,
      "product_id": 24531,
      "label": "Exact"
    },
    {
      "id": 3002,
      "query_id": 27,
      "product_id": 40850,
      "label": "Partial"
    },
    {
      "id": 5322,
      "query_id": 42,
      "product_id": 583,
      "label": "Irrelevant"
    }
  ]
```

1.3.3.1.3 Query Type Analysis

The WANDS dataset includes diverse query patterns that reflect real user search behaviors in e-commerce contexts. To better understand the complexity and variety of user information needs, we categorize the queries into five distinct types based on their characteristics and user intent. Table 1.14 summarizes these query categories with representative examples and their implications for retrieval system design.

This query diversity ensures that retrieval systems must handle various levels of specificity, ambiguity, and user intent, making WANDS a comprehensive evaluation benchmark for e-commerce search applications.

The WANDS dataset includes diverse query patterns that reflect real user search behaviors in e-commerce contexts:

This query diversity ensures that retrieval systems must handle various levels of specificity, ambiguity, and user intent, making WANDS a comprehensive evaluation benchmark.

1.3.3.2 Dataset Setup and Preprocessing

Effective utilization of the WANDS dataset requires systematic setup and preprocessing to prepare data for retrieval system implementation and evaluation.

1.3.3.2.1 Initial Data Loading

The dataset can be accessed and loaded using standard data processing libraries. Download github repo and read CSV files into pandas dataframes:

```
# Clone the WANDS repository
git clone https://github.com/wayfair/WANDS.git

# Load data into pandas dataframes
def get_wands_dfs(wands_folder="../WANDS/dataset/"):
    df_product = get_product_df(wands_folder + "product.csv")
    df_query = get_query_df(wands_folder + "query.csv")
    df_label = get_label_df(wands_folder + "label.csv")

    mask_query = df_label["query_id"].isin(df_query["query_id"])
    mask_product = df_label["product_id"].isin(df_product["product_id"])
    df_label = df_label[mask_query & mask_product]

    return df_product, df_query, df_label
```

Table 1.14 WANDS dataset query type categories and examples.

Query type	Example queries	Characteristics
Categorical queries	"living room furniture," "outdoor dining sets"	Broad category exploration, combining location and product type
Product-specific queries	"l shaped sectional sofa," "round glass coffee table"	Detailed specifications, precise material and shape requirements
Feature-oriented queries	"waterproof outdoor rug," "adjustable height desk"	Functional requirements, mechanism-focused attributes
Intent-based queries	"small space dining solutions," "pet-friendly furniture"	Problem-solving orientation, lifestyle-specific needs
Ambiguous queries	"modern lights," "storage solutions"	Style preferences without specificity, broad functional needs

1.3.3.2.2 Train-test Split Strategy

To ensure reliable evaluation, we implement a stratified split that preserves query and product distribution characteristics:

```
from sklearn.model_selection import train_test_split

# Split queries
df_query_train, df_query_test = train_test_split(
    df_query, test_size=0.25, random_state=42
)

# Create corresponding label splits
df_label_train = df_label[df_label["query_id"].isin(df_query_train["query_id"])]
df_label_test = df_label[df_label["query_id"].isin(df_query_test["query_id"])]
```

This split provides:

- **360 queries (75%)** for training and development of retrieval models.
- **120 queries (25%)** for final evaluation and system comparison.

1.3.3.3 Problem Formulation for Search and Recommendation

The WANDS dataset enables investigation of both search and recommendation scenarios that reflect real-world e-commerce challenges.

1.3.3.3.1 Search Problem Definition

Objective: Given a user search query, retrieve and rank the most relevant products from the catalog to maximize user satisfaction and task completion.

Success Criteria: An effective search system should: (1) prioritize products with "Exact" relevance (score 2) at top positions; (2) appropriately rank "Partial" matches (score 1) in middle positions; (3) minimize or eliminate "Irrelevant" results (score 0) from top rankings; (4) maintain computational efficiency for real-time query processing.

1.3.3.3.2 Recommendation Problem Formulation

Objective: Predict user preferences and suggest relevant products based on implicit signals, behavioral patterns, and contextual information without explicit queries.

Success Criteria: An effective recommendation system should: (1) identify products that align with user interests and needs; (2) balance personalization with diversity to encourage exploration; (3) adapt recommendations based on user context and session behavior; and (4) provide explanations that build user trust and understanding.

Synthetic User Generation: Since WANDS lacks user behavioral data, we create synthetic user profiles. The methods we use include: (1) generate virtual users with diverse preference patterns; (2) assign purchase histories based on product categories and features; (3) create realistic browsing and interaction sequences; and (4) simulate cold-start scenarios for new user recommendations.

Refer to Appendix B for details of the synthetic data generation. This synthetic approach enables comprehensive recommendation system testing while respecting privacy constraints and data availability limitations.

By working through these progressive implementations, readers will gain comprehensive experience with modern IR techniques while developing practical skills directly applicable to real-world e-commerce and recommendation system challenges. The WANDS dataset serves dual purposes as both a hands-on learning tool and a rigorous evaluation benchmark, ensuring that theoretical understanding translates effectively into measurable system improvements.

Throughout the remainder of this book, we will consistently employ the WANDS dataset using the experimental setup and preprocessing procedures established in this section.

1.4 Challenges with IR Systems

While both traditional search and recommendation systems have been successful in various domains, they also face several common challenges:

1.4.1 Lack of Context Awareness

Traditional systems focus heavily on exact term matching or predefined user preferences, often missing the broader context or intent behind a user query. For example, they may struggle to connect a user's interest in "apple fruit" to articles on fruit preservation techniques or health benefits.

1.4.2 Cold-start Problem

In recommendation systems, the cold-start problem arises when there is insufficient user data to make accurate recommendations, particularly for new users or items. This limitation makes it difficult for collaborative filtering models to recommend items to users with limited interaction history.

1.4.3 Scalability

As the volume of data grows, traditional indexing and matching techniques can struggle to scale efficiently, particularly when they rely on exact matching or predefined item attributes (category tags or metadata fields).

1.4.4 Difficulty in Handling Complex Queries

Traditional IR systems are limited in their ability to handle complex or multifaceted queries, such as those requiring a deep understanding of a specific domain or context. For example, a query like "What are the environmental benefits of electric cars in cold climates?" would be difficult for a traditional search engine to resolve with a single result, as it would require an understanding of both environmental science and automotive engineering.

With the advent of LLMs, IR systems have evolved to address many of the shortcomings of traditional approaches. LLMs bring a deeper understanding of language, context, and user intent, allowing for more accurate retrieval of relevant information. By leveraging LLMs, IR systems can now understand just the words used in a query but the intent behind them, providing results that are more relevant and personalized.

In the next sections, we will explore how LLMs enable semantic search and enhance recommendation systems, transforming IR from a basic keyword or feature-matching process into a sophisticated, context-aware retrieval mechanism. Some examples include: enhanced query understanding through contextual intent detection; improved document relevance ranking via semantic similarity scoring; generating concise summaries and answers using summarization and paraphrasing capabilities.

1.5 Summary

This chapter provided a comprehensive foundation in IR, tracing its remarkable evolution from manual card catalogs to sophisticated AI-powered systems. We began by exploring the historical progression through four distinct eras: manual retrieval methods, early computerization featuring Boolean search and vector space models, the web era that introduced PageRank and large-scale indexing, and the current big data era characterized by ML and DL approaches.

We examined the core components of modern IR systems through a three-stage pipeline: pre-retrieval (including user input processing and document indexing), retrieval models (from sparse approaches like TF-IDF and BM25 to dense retrieval using embeddings and neural architectures), and post-retrieval processes (ranking, reranking, and evaluation). This systematic breakdown revealed how traditional keyword-based methods have evolved to incorporate semantic understanding through EBR and hybrid architectures.

The practical applications of IR were illustrated through search and recommendation systems, with hands-on examples using the WANDS e-commerce dataset. These examples demonstrated real-world implementations, including query expansion, hybrid ranking systems, and various recommendation scenarios from next-item prediction to cold-start recommendations.

Despite these advances, we identified persistent challenges in IR systems, including handling ambiguous queries, scaling to massive datasets, and balancing relevance with diversity. These limitations motivate the integration of generative AI capabilities explored in subsequent chapters.

References

1 Jack Delano. Woman at main reading room card catalog in the library of congress, 1930. [Photograph] Between 1930 and 1950. https://www.loc.gov/item/90712184/.
2 Calvin Mooers. Information retrieval viewed as temporal signaling. In *Proceedings of the International Congress of Mathematicians*, volume 1, pages 572–573, 1950.
3 Wikipedia Contributors. Set theory, 2024. https://en.wikipedia.org/wiki/Set_theory. Accessed: 2025-05-25.
4 Gerard Salton and Michael E Lesk. The smart automatic document retrieval system—an illustration. *Communications of the ACM*, 8(6):391–398, 1965.
5 Marcia Bates. *Understanding Information Retrieval Systems*. Auerbach Publications, 2011.
6 Wikipedia Contributors. PageRank — Wikipedia, the free encyclopedia, 2024. https://en.wikipedia.org/wiki/PageRank. Accessed: 2025-5-25.
7 Joseph Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.
8 Terry Winograd. Understanding natural language. *Cognitive Psychology*, 3(1):1–191, 1972.
9 Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2010.
10 Yu A Malkov and Dmitry A Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836, 2018.
11 Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331, 2009.

12 Thorsten Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 133–142, 2002.

13 Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 89–96, 2005.

14 Christopher Burges, Robert Ragno, and Quoc Le. Learning to rank with nonsmooth cost functions. *Advances in Neural Information Processing Systems*, 19:193–200, 2006.

15 Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the 24th International Conference on Machine Learning*, pages 129–136, 2007.

16 Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. Listwise approach to learning to rank: Theory and algorithm. In *Proceedings of the 25th International Conference on Machine Learning*, pages 1192–1199, 2008.

17 Christopher JC Burges. From RankNet to LambdaRank to LambdaMART: An overview. *Learning*, 11(23–581):81, 2010.

18 Andrei Broder. A taxonomy of web search. In *ACM SIGIR Forum*, volume 36, pages 3–10. New York, NY, USA: ACM, 2002.

19 Karen Kukich. Techniques for automatically correcting words in text. *ACM Computing Surveys (CSUR)*, 24(4):377–439, 1992.

20 Hang Cui, Ji-Rong Wen, Jian-Yun Nie, and Wei-Ying Ma. Probabilistic query expansion using query logs. In *Proceedings of the 11th International Conference on World Wide Web*, pages 325–332, 2002.

21 Uma Sawant and Soumen Chakrabarti. Learning joint query interpretation and response ranking. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1099–1110, 2013.

22 Andrew I Schein, Alexandrin Popescul, Lyle H Ungar, and David M Pennock. Methods and metrics for cold-start recommendations. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 253–260, 2002.

23 Eli Pariser. *The Filter Bubble: What the Internet Is Hiding from You*. UK: Penguin, 2011.

24 Carlos A Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):1–19, 2015.

25 Balázs Hidasi and Alexandros Karatzoglou. Recurrent neural networks with top-k gains for session-based recommendations. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, pages 843–852, 2018.