How to become a Staff AI Engineer

Chapter 24: Information Retrieval and Search Engineering for AI Engineering

Celso Sousa
LinkedIn: https://www.linkedin.com/in/celso-sousa/?locale=en-US
Portfolio: https://celsosousa.com.br/project-portfolio/
Ebook: https://celsosousa.com.br/ebook-staff-ai-engineer/

Chapter Objective

Information Retrieval, or IR, is the discipline responsible for finding relevant information within large collections. In modern AI systems, retrieval appears in search engines, RAG, recommendation systems, enterprise search, knowledge assistants, and agentic research.

The problem may appear simple: given a query, find the most relevant items. However, this formulation hides a sequence of architectural decisions. It is necessary to choose mechanisms for lexical retrieval, dense retrieval, approximate search, filtering, fusion, and reranking, while also defining what relevance means, how many candidates need to be retrieved, how much latency can be consumed, and how quality will be measured.

For a Staff AI Engineer, retrieval should not be treated as a simple call to a vector database. Search is a multi-stage system in which different components have distinct objectives, costs, and failure modes.

The objective of this chapter is to develop a mental model for designing retrieval systems while considering end-to-end quality, scale, latency, security, and economics:

- **Candidate generation should prioritize recall:** The first stage must preserve as many potentially relevant items as possible, even if it also retrieves imperfect candidates;
- **Ranking should progressively increase precision:** As the candidate set becomes smaller, it becomes economically feasible to use more sophisticated methods to order the results;
- **Lexical and dense retrieval have complementary strengths:** Exact terms, identifiers, semantic matching, and natural-language queries are not equally well addressed by the same mechanism;
- **Approximation enables vector search to scale:** Exact nearest-neighbor search can become economically infeasible across millions or billions of vectors;
- **Filtering is part of the definition of relevance itself:** Tenant, ACL, language, freshness, and other metadata may determine whether an item is valid regardless of its semantic similarity;
- **Reranking transforms approximate candidates into higher-quality results:** More expensive models can be applied to small candidate sets without being executed across the entire corpus;
- **Evaluation must reflect the product experience:** Recall@K, Precision@K, MRR, MAP, and NDCG measure different properties of the system;
- **Search quality must improve the downstream system:** Better ranking only creates value if it improves task success, answer quality, engagement, resolution, or another relevant outcome.

The primary question is not which search algorithm achieves the best benchmark in isolation. The question is which stage should maximize coverage, which should maximize precision, and how much compute should be invested at each stage to satisfy the product's quality-latency-cost target.

24.1. Search as a Multi-Stage System

A modern search system can be understood as a sequence that progressively reduces the candidate space. A conceptual pipeline can be represented as: **query → candidate generation → filtering → fusion → reranking → final results.**

Each stage either reduces uncertainty or adds a different constraint to the problem:

- **Candidate generation rapidly reduces the search universe:** The objective may be to transform millions or billions of items into hundreds or thousands of candidates;
- **Filtering removes invalid candidates:** Tenant, ACL, language, timestamp, document type, product, or other constraints may prevent specific items from advancing;
- **Fusion combines heterogeneous signals:** Lexical, dense, and other scorers may simultaneously contribute to the candidate set;
- **Reranking applies higher-capacity models:** Because the set has already been substantially reduced, cross-encoders or LLMs can be applied to a limited number of candidates;
- **Final selection incorporates product requirements:** Diversity, freshness, and business rules may modify the ordering produced by previous scorers.

The rationale behind this decomposition is both economic and architectural. Attempting to solve the entire relevance problem in a single stage tends to produce systems that are excessively expensive, slow, or difficult to control. Multi-stage retrieval allows greater computational capacity to be applied only after the candidate set has already been reduced.

This decomposition also makes troubleshooting more precise. A failure in the final results may originate in candidate generation, filters, fusion, the reranker, or final selection. Treating all these behaviors as a single "vector search" problem eliminates this diagnostic capability.

24.1.1. Lexical Retrieval and Inverted Indexes

Lexical retrieval searches for documents based on the terms present in the query. Its fundamental data structure is the inverted index. Instead of representing only **document → terms,** the index maintains a structure conceptually equivalent to: **term → documents containing term.** This makes it possible to quickly locate documents associated with a term without performing a full scan across the entire corpus.

Lexical retrieval remains an extremely important capability even in LLM-based systems:

- **Exact term matching is strong:** Identifiers, codes, names, entities, and technical terminology can be retrieved with high precision;
- **Search is efficient:** The inverted index eliminates a large portion of the corpus before ranking;
- **Behavior is relatively interpretable:** It is possible to analyze which terms contributed to a specific score;
- **Infrastructure is mature:** Search engines can execute lexical retrieval over extremely large collections.

Its primary limitation is semantic mismatch. A query and a document may represent the same information need using different terms. Lexical search may fail to recognize this equivalence, which explains why dense retrieval has become an important complement.

TF-IDF and BM25 represent classical approaches to lexical scoring. The fundamental idea is that not all terms have the same informational value:

- **Term Frequency measures presence within a document:** Repeated occurrence of a specific term may indicate a stronger relationship between the document and the query;
- **Inverse Document Frequency reduces the weight of common terms:** Words that appear throughout almost the entire corpus have low discriminative power;
- **BM25 introduces term-frequency saturation:** Repeating a word hundreds of times does not increase its score indefinitely;
- **BM25 also considers document length:** Normalization reduces distortions associated with document size.

BM25 remains competitive because many workloads depend heavily on exact terminology, named entities, identifiers, and rare keywords. Dense retrieval does not eliminate this requirement.

However, lexical quality depends deeply on how documents and queries are processed. An analyzer pipeline may apply different transformations:

- **Tokenization defines the retrievable units:** An inappropriate tokenizer may destroy codes, URLs, chemical names, or important identifiers;
- **Lowercasing reduces superficial differences:** However, the transformation must be compatible with the domain;
- **Stemming brings word variants closer together:** It may increase recall but can also generate false matches;
- **Stop-word handling reduces certain types of noise:** However, apparently common terms may be semantically relevant in specific domains;
- **Synonym expansion increases lexical coverage:** Aggressive expansion may simultaneously increase recall and decrease precision.

Search configuration is therefore part of the representation layer. High-quality lexical retrieval begins with an accurate understanding of the language and structures of the domain.

24.1.2. Dense Retrieval

Dense retrieval represents queries and documents as embeddings and uses vector proximity to identify semantically related candidates.

Its primary advantage is semantic matching. A relevant document can be retrieved even when its lexical surface differs significantly from the query:

- **Semantic recall may increase:** Conceptual relationships that do not share the same terms become retrievable;
- **Natural-language queries are favored:** Users do not need to know the exact terminology present in the corpus;
- **Cross-lingual retrieval may become possible:** Multilingual embedding models can bring representations from different languages closer together;
- **Cross-modal retrieval may also become possible:** Multimodal embeddings allow distinct modalities to be represented in related spaces.

However, vector similarity does not automatically imply task relevance:

- **Exact identifiers may perform worse:** Small but critical lexical details may be diluted by the semantic representation;
- **Similarity remains only one signal:** A semantically close document may still fail to satisfy the user's actual need;
- **Model choice redefines the vector space:** Replacing the encoder may require re-indexing the corpus;
- **Vector infrastructure introduces new decisions:** ANN algorithms, memory consumption, and index tuning become part of the architecture.

Dense retrieval is particularly strong as a semantic candidate-generation mechanism, but it should not be treated as a universal replacement for lexical retrieval.

24.1.3. Exact Vector Search and Approximate Nearest Neighbor

Exact vector search compares the query against all relevant candidates. This maximizes the probability of finding the true nearest neighbors, but its cost increases with corpus size.

For collections containing millions or billions of vectors, this approach may become excessively expensive. Approximate Nearest Neighbor, or ANN, reduces the explored space while accepting a small probability of missing the exact nearest neighbor.

The objective is to trade a controlled amount of recall for meaningful operational gains:

- **Recall vs. Latency:** Increasing search depth improves the probability of retrieving good neighbors but increases query time;
- **Memory vs. Speed:** Larger auxiliary structures can accelerate queries at the cost of increased memory consumption;
- **Index Build vs. Query Performance:** Sophisticated indexes may provide excellent serving performance while requiring more expensive builds;
- **Freshness vs. Optimization:** Some highly optimized structures are less convenient when updates occur frequently.

Exactness only creates value when its downstream improvement justifies the cost. At large scale, ANN frequently stops being a mere optimization and becomes an economic necessity.

Two groups of techniques illustrate different points within this design space. Hierarchical Navigable Small World, or HNSW, organizes vectors into a navigable graph structure. Search starts at sparser levels and progressively approaches the most promising regions:

- **Search explores only part of the graph:** This avoids comparing the query against every vector;
- **More exploration tends to improve recall:** However, it also increases query latency;
- **The structure may consume substantial memory:** Graph relationships need to be stored;
- **Index construction also has a cost:** Creating high-quality connections requires additional processing.

HNSW tends to be attractive when low latency and high query quality are important and sufficient memory is available. Its tuning should be analyzed through the Recall@K vs. latency vs. memory curve rather than through isolated parameters.

Inverted File Index, or IVF, uses a different strategy. The vector space is partitioned into regions, and the query initially searches the most promising regions.

Detailed search occurs only within those regions:

- **Examining more regions increases recall:** However, it also increases latency;
- **Partitioning reduces the search space:** Poor clusters may hide relevant neighbors;
- **Data distribution influences the index:** Skew may produce regions much larger than others.

Product Quantization, or PQ, adds compression to reduce memory footprint and accelerate certain operations. This is especially important for extremely large corpora, but compression introduces a loss of precision.

HNSW tends to favor query quality and low latency at the cost of a larger memory footprint, whereas IVF/PQ may be attractive when memory density is a critical constraint. None of these technologies is universally superior; the decision depends on the workload and its constraints.

24.2. Hybrid Retrieval, Query Understanding, and Candidate Generation

Lexical and dense retrieval have different failure modes. Hybrid retrieval exploits this complementarity by using more than one mechanism to construct the candidate set.

Consider a query such as: **"XJ-1947 error in recurring payment".** It combines two distinct types of information. The identifier XJ-1947 favors exact matching, whereas recurring payment may benefit from semantic retrieval.

This complementarity explains several benefits of hybrid search:

- **Recall becomes more robust:** Failures from one retriever may be compensated for by another;
- **Heterogeneous queries are handled more effectively:** Natural language, codes, identifiers, and named entities may coexist;
- **Domain shift may be less destructive:** Lexical signals remain useful when the embedding model represents domain-specific terminology poorly.

The new problem becomes combining rankings produced in different score spaces.

24.2.1. Score Normalization and Fusion

BM25 scores and vector similarity do not necessarily have the same scale or distribution. Adding them directly may produce arbitrary results.

Different strategies can address this problem:

- **Score normalization transforms scores into comparable scales:** This enables more controllable weighted combinations;
- **Weighted fusion defines each retriever's contribution:** The system can favor lexical or dense search depending on the workload;
- **Rank-based fusion combines positions instead of scores:** This approach reduces dependence on numerical calibration of the original mechanisms;
- **Reciprocal Rank Fusion combines rankings according to document position:** It can be particularly useful when retrievers produce scores that are difficult to calibrate directly.

However, a single global fusion policy may be inadequate. Identifier-dominated queries and purely semantic queries may require different strategies. Fusion should therefore be evaluated by query segment, and a dynamic retrieval strategy may outperform static weights.

24.2.2. Query Understanding and Transformation

The user's original query is not always the best representation for retrieval. It may be ambiguous, excessively long, conversational, or poorly formulated.

A query transformation layer can produce a representation better suited for retrieval:

- **Query normalization removes noise:** The objective is to preserve intent while eliminating unnecessary variation;
- **Query rewriting makes the information need more explicit:** A conversational formulation can be converted into a query better suited to the corpus;
- **Query expansion adds related terms:** This may increase recall when the query vocabulary is limited;
- **Multi-query retrieval explores multiple formulations:** Different versions may retrieve distinct regions of the corpus.

These techniques can increase coverage, but they introduce a new risk: **semantic drift**. The transformation may alter the original information need.

At Staff level, the problem is not simply determining whether query rewriting improves Recall@K. It is determining whether the system now retrieves the information the user actually wanted more effectively or has started answering a different question.

24.2.3. Metadata Filtering

Relevance does not depend exclusively on document content. In real systems, metadata frequently expresses essential constraints. Examples include tenant, document type, language, timestamp, product, and ACL.

There are two primary points at which these filters can be applied:

- **Pre-filtering restricts the space before search:** Invalid candidates never enter the search path, and certain security guarantees become stronger;
- **Highly selective filters may interact poorly with ANN:** Depending on the index structure, the restriction may significantly reduce the navigable space;
- **Post-filtering performs search before removing invalid candidates:** Implementation may be simpler;
- **Post-filtering may cause candidate starvation:** If 18 of the Top-20 candidates are removed due to authorization, only two useful results remain.

Filtering strategy cannot be designed independently from the index algorithm. Security and retrieval quality also cannot be treated as separate problems.

In particular, authorization must be part of the search path itself. A semantically excellent result belonging to another tenant does not represent low precision; it represents an isolation violation.

24.2.4. Candidate Generation

Candidate generation has a specific responsibility: **rapidly reduce an enormous corpus to a sufficiently small set without discarding relevant items too early.** This stage is primarily recall-oriented.

A candidate generator may use lexical search, dense ANN, hybrid retrieval, or multiple retrievers operating simultaneously:

- **Precision should not be optimized too early:** A relevant document discarded during the first stage cannot be recovered by the reranker;

- **Top-K must preserve sufficient coverage:** A reranker can only operate on candidates it receives;
- **A large K increases downstream cost:** More candidates imply greater scoring and reranking costs;
- **Multiple retrievers may increase coverage:** Different mechanisms may contribute complementary candidates.

Candidate generation should be understood as **coverage under a budget**. The ideal value of K should not be chosen arbitrarily but through the Recall@K vs. downstream cost curve.

This reasoning avoids two extremes. A K that is too small causes irreversible loss of relevant candidates. An excessively large K transfers unnecessary volume to the more expensive components of the pipeline.

24.3. Ranking, Reranking, and Diversity

Once candidate generation has reduced the search space, the objective changes. Instead of prioritizing coverage, the system invests more compute to distinguish similar candidates and improve precision. This difference explains why multi-stage architectures use different models at each stage.

24.3.1. Cross-Encoder Reranking

Bi-encoders represent queries and documents separately. This property allows document embeddings to be precomputed and search to be executed efficiently.

Cross-encoders process the query and document jointly. The interaction between the two texts is richer and frequently produces superior relevance scoring, but its cost per candidate is substantially higher. For this reason, cross-encoders are particularly well suited for reranking.

A conceptual pipeline may:

1. retrieve 100 candidates;
2. score the 100 candidates with a cross-encoder;
3. return the Top-10.

This pattern concentrates compute where it provides the greatest leverage:

- **Query-document interaction is richer:** The model directly evaluates the relationship between the two contents;
- **Hard negatives can be distinguished more effectively:** Documents that are semantically close but incorrect can receive different scores;
- **Quality increases without executing the expensive model across the entire corpus:** Candidate generation preserves scalability.

The primary trade-off is latency. Reranking sits on the query's critical path and must remain compatible with the product's latency budget.

24.3.2. LLM Reranking

LLMs can also be used as rerankers when the definition of relevance involves more complex criteria, nuances, or sophisticated instructions.

This flexibility comes at a significant cost:

- **Latency may increase substantially:** Inference must be applied to the candidates being compared;
- **Serving cost increases:** Ranking begins consuming resources associated with LLM execution;
- **Nondeterminism increases variance:** Rankings may change across executions;
- **Prompt sensitivity begins influencing retrieval:** The reranker itself becomes dependent on another Prompt Engineering layer;
- **Context limits restrict the number of candidates:** Many documents may not fit simultaneously within the context window;
- **Apparent explainability can be misleading:** A plausible textual justification does not guarantee that the ranking is correct.

LLM reranking should be used when the semantic complexity of ranking justifies its cost. Replacing an efficient cross-encoder with an LLM solely because it appears more sophisticated does not necessarily represent an engineering improvement.

The criterion should be incremental value: **how much the new reranker improves the experience or downstream outcome and how much latency and cost it adds to produce that improvement.**

24.3.3. Diversity-Aware Ranking

Even an accurate ranking may produce a poor final result set. The Top-10 documents, for example, may essentially represent the same information.

Diversity-aware ranking adds another dimension to the problem:

- **Relevance remains necessary:** Results need to answer the query;
- **Novelty reduces repetition:** Additional items should contribute new information;
- **Coverage broadens evidence:** Different sub-aspects of the information need can be represented.

This strategy can be important in RAG, recommendation, and exploratory search. In RAG, redundant documents consume context budget without adding evidence. Diversification can reduce repetition, increase perspective coverage, and decrease token waste.

However, diversity also has a cost. Forcing variety may introduce documents that are individually less relevant. The architecture must determine how much relevance it is willing to sacrifice in exchange for a more informative set.

24.4. Retrieval and Ranking Evaluation

Search cannot be reliably improved without proper evaluation. However, no single metric represents every property of a retrieval system.

The first important distinction is between precision and recall:

- **Precision measures the proportion of retrieved items that are relevant:** It becomes especially important when returning incorrect results has a high cost;
- **Recall measures the proportion of relevant items that were retrieved:** It is critical when missing relevant information represents the primary risk;
- **Recall@K is particularly important in candidate generation:** The first stage needs to preserve candidates for subsequent stages;
- **Precision@K becomes more important in final ranking:** Results presented to the user need to concentrate relevance.

This explains a common RAG architecture: high-recall candidate generation followed by a reranker oriented toward higher precision.

There is no "best metric" outside context. The correct question is which type of error has the greatest cost at that specific stage of the pipeline.

24.4.1. Position-Based Metrics

Precision and Recall do not adequately capture situations in which the result position has a strong impact. MRR, MAP, and NDCG represent different ranking properties:

- **Mean Reciprocal Rank prioritizes the first relevant result:** It is particularly appropriate when the user typically needs to find a single correct item, such as in answer retrieval or navigation;
- **Mean Average Precision considers multiple relevant results:** It is useful when multiple correct items can contribute to the experience;
- **Normalized Discounted Cumulative Gain supports graded relevance:** Highly relevant items positioned near the top contribute more, making NDCG appropriate for many ranking, recommendation, and search problems.

The metric must reflect actual user behavior. If only the first result is typically consumed, a metric that distributes value across the entire ranking may obscure the most important property of the experience.

24.4.2. Offline Evaluation

Offline evaluation allows search configurations to be compared before they are deployed to production. This requires query sets and relevance judgments.

The quality of the evaluation set determines the usefulness of the benchmark:

- **Queries must be representative:** Language and intent should reflect real traffic;
- **Hard negatives must be included:** Plausible but incorrect candidates expose real differences between ranking strategies;
- **Long-tail cases must be represented:** Rare queries may have far greater business importance than their frequency suggests;
- **Results must be segmented:** Language, domain, query type, and difficulty may hide very different behaviors when aggregated.

An important risk is benchmark overfitting. If all optimizations are continuously performed against the same set, the search system may learn to improve the benchmark rather than the product. Mature evaluation must preserve holdouts, refresh queries, and compare offline results against production behavior.

24.4.3. Online Evaluation and Business Metrics

Offline metrics are proxies. Production reveals the actual impact of a change.

Search quality may alter click-through rate, successful resolution, downstream answer accuracy, or task completion. In RAG systems, better retrieval may also reduce hallucination or inappropriate abstention.

However, online metrics require interpretation. A ranking that increases clicks has not necessarily improved relevance; it may simply be promoting results with more attractive titles.

Evaluation should therefore connect multiple layers:

- **Retrieval metrics expose search behavior:** Recall@K, Precision@K, MRR, and NDCG help localize ranking improvements;
- **System quality measures downstream impact:** In RAG, for example, answer correctness or hallucination may reveal whether better candidates actually improved generation;
- **Business outcomes measure final value:** Task completion, resolution, engagement, or other metrics indicate whether the change produced an effect on the product.

Search must be evaluated as part of the complete system. Improving NDCG without improving the final task may indicate that the optimized component was not the actual bottleneck.

24.5. Search in Production: Freshness, Distribution, Security, and Economics

A retrieval system does not end when the index is built. In production, documents are continuously added, modified, and removed. The system must also handle distributed scale, authorization, attacks, observability, and operational cost. These dimensions modify the definition of search quality itself.

24.5.1. Freshness and Consistency

Search systems operate over mutable information. New documents arrive, old content changes, and items cease to exist.

This introduces several operational properties:

- **Indexing latency measures how long a change takes to become searchable:** An index that is correct but delayed may continue returning obsolete knowledge;
- **Update semantics determine how versions are replaced:** Modified documents need to correctly replace previous representations;
- **Deletion correctness guarantees effective removal:** Removed content should not remain accessible through indexes or caches;
- **Version awareness prevents results that are semantically correct but operationally outdated:** A new policy may make an earlier version inappropriate.

Freshness may be an integral component of relevance. An old document can have perfect semantic similarity and still be the operationally wrong answer.

Search quality does not merely mean retrieving the correct content. It means retrieving the **right content, right version, right time**.

24.5.2. Distributed Search

At large scale, the corpus may need to be distributed across shards. Each shard retrieves its best candidates, and a subsequent stage performs the global merge.

This architecture creates new trade-offs:

- **More shards increase parallelism:** But they also increase fan-out and coordination overhead;

- **Skew can create hotspots:** Some shards may concentrate more documents or traffic and determine tail latency;
- **Distributed Top-K requires a global merge:** Local results need to be compared across partitions;
- **Replication improves availability and read capacity:** However, it increases storage consumption.

End-to-end latency often stops being determined by the average and instead depends on the slowest shard. Partitioning, skew, and p99 must therefore be analyzed together.

24.5.3. Security

Retrieval is also a security boundary. Semantic relevance must never override authorization.

The primary risks need to be addressed within the search design itself:

- **ACLs must be enforced before exposure:** An unauthorized document must not reach either the user or the LLM;
- **Tenant isolation must remain in the retrieval path:** Retrieving a result belonging to another tenant is a security incident;
- **Index poisoning can manipulate ranking:** Content may be deliberately created to appear among the results;
- **Query abuse can cause resource exhaustion:** Broad queries, traversals, or expensive filters require limits;
- **Search logs may contain sensitive information:** Observability must follow the same privacy concerns as the rest of the system;
- **Metadata can also leak information:** The retrieval layer must not improperly reveal the existence of protected content through results, scores, or metadata.

Security cannot be added after retrieval. Once unauthorized content has been exposed to a model or user, correcting the ranking does not undo the exposure.

24.5.4. Economics

Search incurs indexing, storage, memory, compute, and reranking costs. As the system grows, small pipeline decisions can have significant economic impact.

Several principles help control these costs:

- **Candidate pruning reduces downstream cost:** Fewer documents reach the more expensive stages;
- **Caching can reduce work for repeated queries:** Provided that freshness and authorization allow safe reuse;
- **Higher K simultaneously increases quality and cost:** The economically optimal point needs to be measured;
- **Reranking must demonstrate incremental value:** An expensive model that improves NDCG only marginally may not justify the serving burden;
- **Index memory must be treated as an economic resource:** Structures with excellent latency may consume substantial amounts of memory;
- **Hybrid search improves robustness but also increases infrastructure requirements:** Maintaining multiple retrieval paths has an operational cost.

The most useful economic metric is not necessarily cost per query in isolation. When possible, the objective should be to understand cost per successful retrieval or per successful downstream task.

24.5.5. Observability

A multi-stage pipeline needs to expose telemetry for each stage. Without this visibility, an end-to-end degradation may be incorrectly attributed to the model, index, or reranker.

Observability should make it possible to answer questions such as:

- **Latency by stage:** How much time is consumed by candidate generation, filtering, fusion, and reranking?;
- **Candidate count:** How many candidates enter and leave each stage?;
- **Zero-result rate:** How many queries fail to produce usable results?;
- **Quality by query class:** Which segments have lower Recall?;
- **Drift:** Did quality change after modifying index configuration, the embedding model, or the corpus?;
- **Reproducibility:** Can the ranking produced for a historical query be reconstructed?

This visibility is essential for distinguishing representation, indexing, retrieval, authorization, ranking, and serving problems.

24.6. Trade-Offs and Anti-Patterns

Information Retrieval can be understood as a discipline for managing trade-offs among coverage, precision, latency, memory, security, and cost. There is no universally optimal configuration.

The major trade-offs appear repeatedly:

- **Lexical vs. Semantic Retrieval:** Exact matching preserves specific terms, whereas dense retrieval captures conceptual equivalence;
- **Exact Search vs. ANN:** Exact search maximizes nearest-neighbor accuracy, whereas ANN makes large-scale retrieval economically viable;
- **Recall vs. Latency:** Exploring more candidates tends to increase recall while consuming more compute;
- **Candidate K vs. Reranking Cost:** A larger K preserves more potentially relevant items but transfers more volume to expensive stages;
- **Precision vs. Recall:** More conservative results reduce noise but may miss necessary information;
- **Filtering vs. Candidate Availability:** Strong constraints increase validity but may excessively reduce the candidate set;
- **Relevance vs. Diversity:** Individually more relevant items may be redundant, whereas greater diversity may introduce lower-scoring results;
- **Freshness vs. Index Optimization:** Highly optimized indexes may be more difficult to update rapidly;
- **Search Quality vs. Infrastructure Cost:** Larger indexes and more sophisticated rerankers need to justify their downstream impact;
- **Memory vs. Query Performance:** Structures such as HNSW may trade a larger footprint for low latency;
- **Static Fusion vs. Query-Specific Strategy:** Global weights simplify the architecture, whereas query segmentation can better capture heterogeneous workloads.

Staff-level work consists of converting these trade-offs into measurable decisions. The selected technology must satisfy the properties that matter to the workload rather than merely produce good results on a generic benchmark.

Several anti-patterns are particularly important:

- **Using only dense retrieval because the system includes an LLM:** Identifiers, rare terms, and exact terminology remain relevant;
- **Using only BM25 for highly semantic workloads:** Lexical mismatch may significantly reduce recall;
- **Treating similarity as absolute relevance:** Vector distance represents only one signal;
- **Adding BM25 and vector scores without calibration:** Incompatible scales can make fusion arbitrary;
- **Increasing Top-K indefinitely to improve recall:** Reranking latency, compute, and context pollution also increase;
- **Running an expensive reranker across the entire corpus:** This eliminates the economic advantage of a multi-stage architecture;
- **Evaluating only easy queries:** Hard negatives expose the true capability of the ranking system more clearly;
- **Ignoring authorization during retrieval:** Security cannot be repaired after improper information has already been exposed;
- **Using LLM reranking without a cross-encoder baseline:** Greater complexity must demonstrate measurable gains;
- **Optimizing only one offline metric:** Better NDCG does not guarantee better task success;
- **Ignoring freshness:** The semantically closest document may be an obsolete version;
- **Distributing shards without analyzing skew:** Hotspots, fan-out, and tail latency may dominate system performance.

These anti-patterns share a common characteristic: **optimizing an isolated component without considering its end-to-end consequences.**

24.7. Practical Framework for Designing Search and Retrieval

A Staff AI Engineer can use a relatively systematic sequence of analysis when designing or reviewing a retrieval system. First, determine what relevance means for the product:

- **Define the user's need:** Determine whether the user is searching for a single item or multiple relevant results;
- **Evaluate exact terminology:** Identify whether codes, names, or identifiers are essential;
- **Determine the role of semantics:** Verify whether semantic similarity solves a significant portion of queries;
- **Include freshness when necessary:** More recent content may be preferable even with a lower semantic score;
- **Consider source authority:** In some systems, the origin of the document may affect its validity;
- **Define permission constraints:** Some results must never be returned;
- **Quantify the cost of errors:** Missing a relevant document and returning an incorrect document may have very different consequences.

This definition guides every subsequent stage. Next, characterize the actual queries:

- **Analyze format and length:** Queries may be short, structured, or conversational;
- **Identify identifiers and exact names:** These segments frequently favor lexical retrieval;
- **Compare query language and corpus language:** Terminological differences may increase the importance of semantic retrieval;
- **Identify ambiguity:** Some queries may require additional interpretation;
- **Evaluate query rewriting and expansion:** These techniques should remain only if they improve quality without introducing semantic drift;

- **Segment query classes:** Different workloads may justify different strategies.

Next, build a strong lexical baseline:

- **Evaluate BM25 before adding complexity:** The baseline may be competitive across many workloads;
- **Analyze discriminative terms:** Certain fields or tokens may concentrate a large portion of the value;
- **Review analyzers:** Domain terminology must be preserved;
- **Measure the impact of stemming:** Recall gains need to be compared against false positives;
- **Test synonym expansion:** Expansion is useful only when it increases coverage in a controlled manner;
- **Validate identifier tokenization:** Codes and structured terms must not be destroyed;
- **Consider field weighting:** Different fields may contribute differently to relevance.

Next, evaluate dense retrieval:

- **Test the embedding model within the actual domain:** Generic quality does not guarantee domain-specific retrieval quality;
- **Verify compatibility between query and document representations:** Encodings need to operate according to the model's expectations;
- **Use the appropriate similarity metric:** Configuration must respect the embedding model being used;
- **Determine whether exact vector search is still viable:** ANN does not need to be introduced before scale requires it;
- **Analyze false semantic neighbors:** Semantically close results may still be irrelevant to the task;
- **Preserve lexical fallback when necessary:** Dense search does not need to assume responsibility for every query type.

When exact vector search is no longer appropriate, evaluate ANN:

- **Design for the current and future corpus:** Index choice must account for growth;
- **Define the minimum acceptable Recall@K:** Approximation must remain compatible with downstream quality;
- **Establish the latency budget:** Search depth must fit within the critical path;
- **Determine the memory budget:** Index configuration must respect available resources;
- **Consider update frequency:** Freshness requirements affect structure selection;
- **Compare HNSW and IVF/PQ on the actual workload:** The decision should reflect measured curves rather than technological preference;
- **Build Recall vs. latency curves:** Configurations should be compared according to the frontier they provide.

Next, determine whether hybrid retrieval actually adds value:

- **Identify queries for which BM25 is superior:** Exact terms and identifiers may reveal clear segments;
- **Identify queries for which dense retrieval is superior:** Semantic mismatch may reveal another segment;
- **Measure the gain from the combination:** Duplicating infrastructure without measurable gains does not create value;
- **Define score normalization where required:** Incompatible scores must not be combined arbitrarily;
- **Test weighted fusion:** Weights need to remain stable across query types;
- **Compare against rank fusion:** RRF may be more robust when scores are difficult to calibrate;
- **Evaluate dynamic strategies:** Query-specific routing may outperform global weights.

Next, design filtering jointly with the index:

- **List mandatory metadata constraints:** Tenant, ACL, language, timestamp, and other filters need to be explicit;
- **Guarantee tenant and ACL enforcement in the search path:** Authorization should not depend exclusively on a later stage;
- **Test pre-filtering efficiency:** Some ANN indexes respond differently to selective filters;
- **Measure candidate starvation under post-filtering:** The nominal candidate count may hide a very small number of valid results;
- **Adapt the strategy for highly selective filters:** A single configuration may not work for every segment;
- **Validate the non-exposure guarantee:** Unauthorized items must never reach either the user or the model.

Next, size candidate generation:

- **Choose K based on Recall:** The candidate set needs to preserve relevant items;
- **Measure the marginal gain from increasing K:** Every additional candidate has downstream cost;
- **Quantify reranking cost:** Candidate budget and inference budget are connected;
- **Analyze the contribution of multiple retrievers:** Each mechanism should add useful coverage;
- **Measure redundancy:** Many equivalent candidates waste budget;
- **Verify losses before the reranker:** An excellent reranker cannot recover documents that were never retrieved.

Next, determine whether reranking is necessary:

- **Measure the quality of first-stage ranking:** Additional complexity is only necessary if there is meaningful room for improvement;
- **Evaluate cross-encoders using appropriate metrics:** Gains in NDCG, MRR, or another metric need to be material;
- **Choose how many candidates will be reranked:** Quality and latency need to be balanced;
- **Confirm compatibility with the latency budget:** Reranking must not make the experience operationally infeasible;
- **Evaluate batching where applicable:** Better compute utilization may reduce serving burden;
- **Test LLM reranking only against a competitive baseline:** Additional gains need to compensate for cost and nondeterminism;
- **Compare benchmark results against production:** Better offline performance must also appear in the real system;
- **Look for cheaper alternatives:** Similar quality with lower complexity may represent a better architecture.

Next, analyze diversity:

- **Measure redundancy in the Top-K:** Highly similar results may waste candidate or context budget;
- **Evaluate the need for complementary evidence:** RAG may benefit from distinct perspectives;
- **Quantify repeated information:** Many chunks may express essentially the same content;
- **Measure the impact on answer quality:** Diversification must produce a better downstream result;
- **Quantify the cost in individual relevance:** Additional coverage is not free.

Next, define offline evaluation:

- **Use production or representative queries:** Golden queries need to reflect real behavior;
- **Ensure the quality of relevance judgments:** Poor ground truth produces poor evaluation;
- **Include hard negatives:** Plausible but incorrect cases expose differences between systems;
- **Choose metrics according to the stage:** Recall@K, MRR, and NDCG answer different questions;

- **Segment evaluation:** Query type, language, domain, and difficulty may exhibit different behaviors;
- **Preserve an independent holdout:** This reduces benchmark overfitting.

Next, connect retrieval to the downstream system:

- **Verify whether higher Recall@K improves answer correctness:** More candidates do not automatically imply better answers;
- **Measure whether higher NDCG reduces hallucination:** Ranking only creates value when it influences the task;
- **Observe task success:** Search changes need to produce end-to-end improvement;
- **Measure user effort:** Good results should make information easier to find;
- **Look for divergent segments:** Retrieval metrics may improve while product metrics deteriorate;
- **Optimize the final outcome:** The architecture exists to improve the product, not ranking in isolation.

Next, validate freshness and consistency:

- **Define acceptable indexing latency:** The time between source update and index update must be explicit;
- **Validate document replacement:** Updates must not leave old versions competing with new ones;
- **Guarantee deletion correctness:** Deleted content must disappear from indexes and caches;
- **Consider version metadata in ranking:** Certain applications need to favor valid versions;
- **Monitor stale results:** Old content appearing among the top results may represent a severe failure;
- **Use freshness as a ranking signal when necessary:** Recency may be part of relevance itself.

Next, analyze distribution and scale:

- **Size the number of shards:** Parallelism must justify coordination overhead;
- **Choose a partitioning strategy:** Distribution affects balancing and query execution;
- **Measure skew:** Imbalanced shards may dominate p99;
- **Quantify fan-out:** Each query may multiply work across the cluster;
- **Measure the cost of Distributed Top-K:** Global merge also adds latency;
- **Define replication according to actual requirements:** Availability and throughput must justify additional storage;
- **Identify the shard that determines tail latency:** The system needs to be analyzed by the tail, not only the average;
- **Detect full-cluster queries:** Some queries may eliminate the benefits of partitioning.

Next, review security:

- **Guarantee ACL enforcement before exposure:** Unauthorized information must never reach the LLM or the user;
- **Validate tenant isolation:** Separation must be embedded into the retrieval path;
- **Protect search logs:** Queries may contain sensitive information;
- **Consider adversarial content:** Documents may be deliberately created to manipulate ranking;
- **Protect against index poisoning:** Malicious content may alter retrieval behavior;
- **Limit expensive queries:** Quotas, timeouts, and candidate limits reduce resource exhaustion;
- **Prevent side-channel leakage:** Metadata or scores must not improperly reveal the existence of protected data.

Next, ensure sufficient observability:

- **Instrument latency by stage:** Candidate generation, filtering, and reranking need to be distinguishable;

- **Record candidate counts:** Reduction of the search space must be observable;
- **Monitor zero-result queries:** They may indicate coverage or filtering problems;
- **Segment quality:** Specific classes may perform far below the average;
- **Detect drift:** Changes to the index, encoder, or corpus need to be correlated with quality changes;
- **Preserve ranking reproducibility when necessary:** Investigating historical queries requires knowing which configuration was active at that time.

Next, evaluate economics:

- **Calculate cost per query:** This is an important operational baseline;
- **Measure the index memory footprint:** Search latency may be purchased with expensive infrastructure;
- **Calculate reranking cost per candidate:** This directly connects candidate K to serving cost;
- **Measure the incremental value of K:** Additional candidates should remain only when their quality gain justifies their cost;
- **Quantify the additional cost of hybrid retrieval:** Robustness must be compared against additional infrastructure;
- **Evaluate caching safely:** Freshness and authorization constrain reuse;
- **Compare smaller rerankers:** Cheaper models may provide nearly equivalent quality;
- **Track cost per successful retrieval or task:** Cost needs to be connected to the outcome produced.

Finally, connect the architecture to the business:

- **Measure impact on task completion:** Better search needs to enable users or systems to complete tasks more successfully;
- **Measure reduction in human effort:** Retrieval can create value by reducing manual work;
- **Measure impact on hallucination:** In RAG, search directly contributes to answer quality;
- **Connect quality to conversion, engagement, or resolution rate when applicable:** Technical metrics need to reach the product level;
- **Determine how much search quality is actually necessary:** Not every workload needs the most sophisticated ranking possible;
- **Avoid paying for recall that is not used downstream:** Excess capacity also represents waste;
- **Evaluate architectural reuse:** A search stack may or may not justify centralization across multiple products;
- **Question expensive stages:** Removing a component should be evaluated according to the business value actually lost.

This sequence represents the mindset expected from a Staff AI Engineer. Search should not be treated as an isolated call to BM25 or a vector database. It is a multi-stage system in which candidate generation, filtering, fusion, and reranking have distinct responsibilities, explicit budgets, and their own metrics.

24.8. Chapter Summary

Modern Information Retrieval is a systems engineering discipline. Final quality emerges from the interaction among representation, indexing, retrieval, filtering, ranking, evaluation, and operations.

The core principles can be summarized as follows:

- **Use lexical retrieval for exact signals:** Terms, identifiers, and named entities remain essential even in LLM-based architectures;

- **Use dense retrieval for semantic coverage:** Embeddings help when query language differs from corpus language;
- **Combine both when the workload is heterogeneous:** Hybrid retrieval may produce more robust behavior;
- **Use ANN when exact vector search no longer scales:** Recall needs to be balanced against latency, compute, and memory;
- **Choose HNSW, IVF, or PQ according to the workload:** There is no universally superior vector index;
- **Separate candidate generation from reranking:** First preserve coverage, then invest compute to increase precision;
- **Use cross-encoders when query-document interaction justifies the cost:** Small candidate sets make this approach economically feasible;
- **Use LLM reranking only when there is measurable gain:** Sophistication is not a substitute for evidence;
- **Include filtering in retrieval design:** Metadata, ACLs, and freshness are part of relevance itself;
- **Choose metrics according to the experience:** Recall@K, Precision@K, MRR, MAP, and NDCG represent different properties;
- **Evaluate with hard negatives and representative queries:** Easy benchmarks may significantly overestimate search quality;
- **Connect search metrics to downstream outcomes:** Better retrieval must produce better behavior in the complete system;
- **Monitor freshness and consistency:** Correct search must return the correct information, in the correct version, at the correct time;
- **Treat retrieval as a security boundary:** Semantic relevance must never override authorization;
- **Optimize quality per unit of latency and cost:** The best search system is not the one that executes the most sophisticated components, but the one that delivers appropriate candidates within the product's budgets.

The primary lesson is that Information Retrieval is not a single search operation. It is a pipeline of decisions that progressively reduces the candidate space while combining lexical signals, vector similarity, metadata, filtering, fusion, and reranking.

At Staff level, the question stops being "which search technology should we use?" and becomes: **which architecture preserves the required candidates, ranks them with sufficient quality, and delivers the desired downstream outcome within the constraints of latency, memory, security, operations, and cost?**

This shift in perspective transforms retrieval from an infrastructure component into a central discipline of AI Engineering.