DESIGNING HYBRID SEARCH SYSTEMS

László Csontos

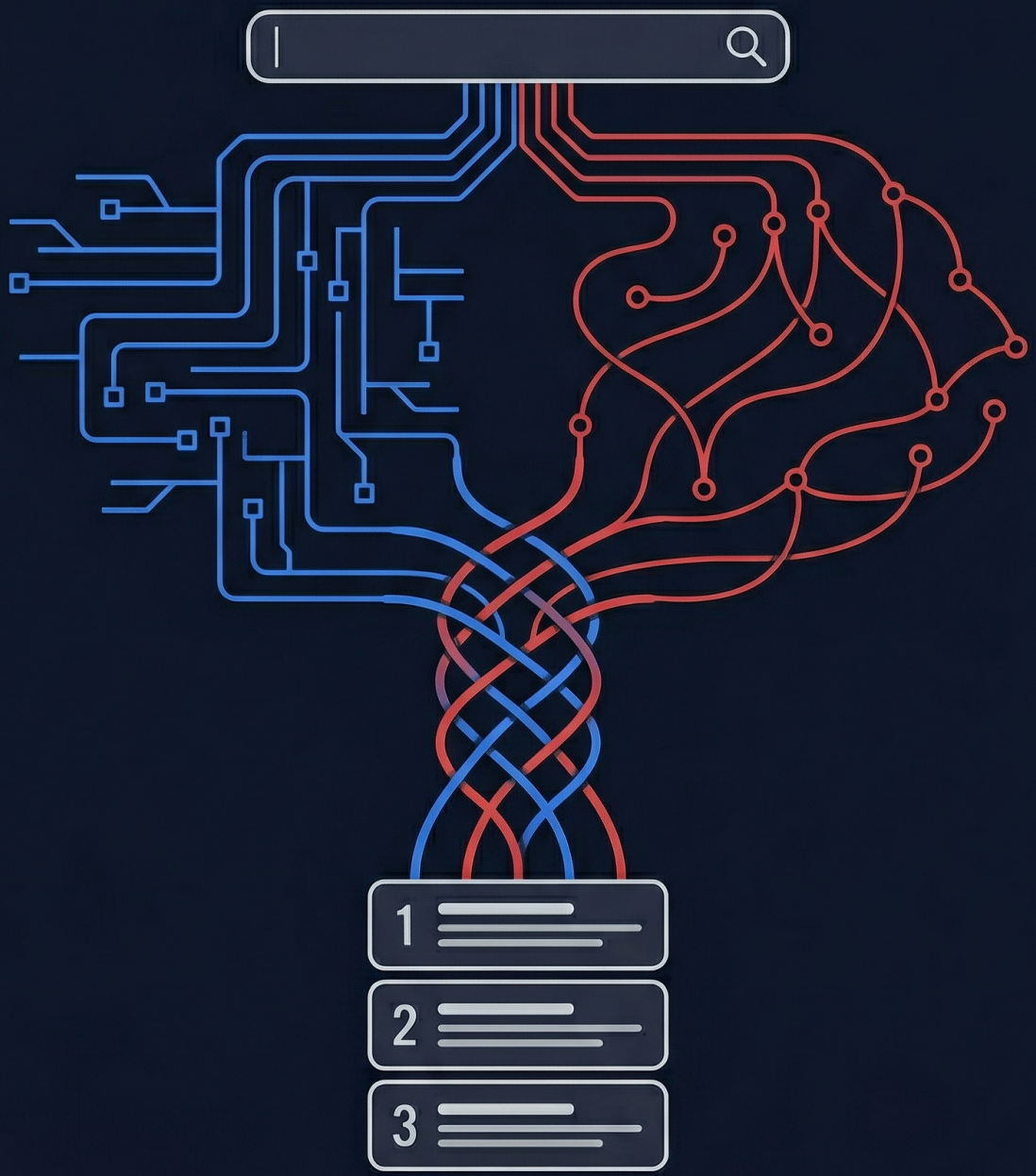

Table of contents

Introduction

Every search system makes a promise: give me your question, and I will find the answer. Most systems break that promise dozens of times a day, and users notice. They just stop searching and start browsing, or they leave.

The gap between what users expect from search and what most production systems deliver has widened over the past decade. Consumer search behavior has been shaped by Google, and users now bring those expectations to every search box they encounter. When those expectations are not met, the consequences are measurable: 94% of consumers report receiving irrelevant results on retail websites, and 87% (on average across the US, Brazil, India, Mexico, Australia, and the UK) say those experiences changed how they viewed the brand [1]. They type natural language questions into e-commerce catalogs. They paste entire paragraphs into internal knowledge bases. They expect a support portal to understand what they mean, not just what they typed. Meanwhile, the search systems behind those boxes are often running the same inverted index configurations they were built with years ago.

Two waves of technology have tried to close this gap, and both have fallen short on their own.

The first wave, lexical search, gave us BM25, TF-IDF, and inverted indexes. These systems are fast, predictable, and well-understood. They excel at exact match, structured filtering, and queries where the user knows the precise vocabulary of the domain. But they fail silently when a user searches for “lightweight laptop for travel” and the product

description says "ultraportable notebook." The vocabulary mismatch problem is not an edge case. Usability research on e-commerce search has documented that 41% of e-commerce search engines perform below users' expectation [2]. Every synonym, abbreviation, and colloquial phrasing that doesn't match the indexed text is a missed result.

The second wave, vector search, promised to solve this by encoding meaning rather than matching words. Embedding models map queries and documents into a shared vector space where semantic similarity can be measured directly. A search for "lightweight laptop for travel" now lands near "ultraportable notebook" because their meanings are close, regardless of lexical overlap. This works remarkably well for many queries. It also fails in ways that lexical search never did. A user searching for "Apple stock price" may receive results about apple nutrition facts, because the embedding model maps both uses of "apple" to nearby regions of vector space. More broadly, vector search struggles with exact entity matching ("model XPS-13-9340"), with negation ("laptops without touchscreen"), with numerical precision ("under $800"), and with low-frequency domain-specific terms that embedding models never saw during training. Worse, these failures are harder to diagnose. A keyword search that returns no results is an obvious signal. A vector search that returns confidently wrong results looks like it's working.

Neither approach is sufficient. Both are necessary. That is the core premise of this book.

What Hybrid Search Actually Means

Hybrid search combines lexical and semantic retrieval into a single pipeline. At its simplest, this means running a BM25 query and a vector similarity query in parallel, then merging their results using a fusion algorithm such as Reciprocal Rank Fusion (RRF) or a learned scoring function. In practice, production hybrid search systems are

substantially more complex. They include query understanding layers that classify intent and extract entities before retrieval begins. They include reranking stages where cross-encoder models rescore the merged candidate set for fine-grained relevance. They include feedback loops where user behavior data flows back into model training and relevance tuning.

The term "hybrid search" has moved from research concept to production infrastructure faster than most practitioners realize. Every major search platform now supports it as a first-class feature. Elasticsearch, OpenSearch, Pinecone, Weaviate, Qdrant, Milvus, Redis, and pgvector all expose hybrid query APIs. Cloud providers offer managed hybrid search through AWS OpenSearch Serverless, Google Vertex AI Search, and Azure AI Search. MongoDB introduced a dedicated `$rankFusion` operator for hybrid search in version 8.0. Microsoft built an `IKeywordHybridSearchable` interface into Semantic Kernel. The term is embedded in production APIs across the industry, which means it is not going away.

This rapid adoption has created a problem. The tooling exists, but the architectural knowledge to use it well does not. Engineers can turn on hybrid search with a few API calls. Knowing when to use it, how to tune the fusion weights, which embedding model to choose for a given domain, how to evaluate whether it actually improved relevance, and how to operate it at scale without blowing up the infrastructure budget: that knowledge is scattered across vendor documentation, conference talks, blog posts, and tribal expertise inside a handful of companies.

This book exists to consolidate that knowledge into one place.

Who This Book Is For

This book is written for four overlapping audiences.

Search and platform engineers building or maintaining search infrastructure will find the architectural patterns, scaling strategies, and operational guidance they need to design hybrid search systems that hold up in production. The book assumes familiarity with building software systems but does not assume prior search expertise.

ML engineers and RAG team leads will find practical guidance on embedding model selection, fine-tuning, reranker training, and evaluation methodology. The retrieval component of a RAG pipeline is often the weakest link, and this book treats retrieval as the primary engineering challenge rather than an afterthought between the LLM and the user.

Engineering managers and CTOs evaluating whether to invest in hybrid search, choosing between build and buy, or planning a migration from legacy keyword search will find decision frameworks, vendor comparisons, and cost analysis throughout the book. Chapters 3, 7, and 17 are written with this audience in mind.

Enterprise architects designing knowledge search systems across heterogeneous document corpora, with access control, compliance requirements, and integration constraints, will find dedicated coverage in Chapter 20.

The book is not a textbook. It does not derive algorithms from first principles or prove convergence theorems. Appendix A provides a mathematical quick reference for readers who want the formulas behind BM25, cosine similarity, RRF, and HNSW, but the main text focuses on system design decisions and their consequences. Readers looking for a rigorous mathematical treatment of information retrieval should start with *Manning, Raghavan, Schütze: Introduction to Information Retrieval* [3], which remains the best foundation. This book picks up where that one leaves off: at the boundary between theory and production.

Why This Book, Why Now

Three market shifts have converged to make hybrid search a distinct engineering discipline rather than a niche technique.

The first shift is the explosion of retrieval-augmented generation (RAG). The RAG market reached $2.33 billion in 2025 and is projected to grow at a 42.7% compound annual rate through 2035 [4]. Every RAG pipeline needs a retrieval layer, and the quality of that retrieval layer determines whether the LLM generates useful answers or plausible nonsense. Organizations that invested heavily in fine-tuning LLMs are discovering that retrieval is the highest-leverage component in the pipeline: if the retrieval layer fails to surface the right documents, no amount of prompt engineering or model improvement can compensate. Hybrid search is the default retrieval strategy for production RAG systems, and the engineering teams building these systems need architectural guidance that doesn't exist in book form today.

The second shift is the maturation of vector search infrastructure. Two years ago, running vector search at scale required significant custom engineering. Today, the vector database market has reached $2.55 billion with a projected 22.3% compound annual growth rate through 2035 [5]. Purpose-built vector databases, hybrid-capable search engines, and managed cloud services have made the infrastructure accessible. The bottleneck has moved from "can we run vector search?" to "how do we design the system around it correctly?"

The third shift is the recognition that search quality directly impacts revenue and operational efficiency. Target improved product discovery relevance by 20% after rebuilding its search platform around hybrid retrieval with AlloyDB AI [6]. Stack Overflow migrated from pure lexical search to a hybrid Elasticsearch and semantic search architecture to handle the complexity of developer queries, where users need both semantic understanding of problem descriptions and exact matching of error messages and code snippets [7]. These are not experimental deployments. They are production systems serving millions of users,

and the organizations behind them measured the business impact carefully.

Despite all of this, no book covers hybrid search system design as its primary subject. Existing titles address hybrid search as one topic among many (within broader search, RAG, or platform-specific coverage), but none treats the end-to-end system design problem as the central focus. This book fills that gap. It is the first dedicated treatment of hybrid search as a system design problem.

How This Book Is Organized

The book is structured in six parts. Each part is designed to be useful on its own, so readers can enter at whatever level matches their current needs.

Part I: Why Hybrid Search (Chapters 1-3) frames the problem. It walks through the failure modes of pure keyword search and pure vector search, then makes the case for hybrid retrieval with a decision framework for choosing a strategy. Readers evaluating whether hybrid search is worth the investment, or building the case for a migration, should start here. Readers already convinced that hybrid search is the right approach can skip to Part II.

Part II: Architecture (Chapters 4-7) covers the system design core. It presents reference architectures for hybrid search pipelines, including query routing that selects the right retrieval path per query type. It then dives into query understanding as an architectural layer, explains the reranking stage, and provides a vendor-neutral comparison of search platforms. Engineers designing a new search system or re-architecting an existing one should start here.

Part III: Models (Chapters 8-10) addresses model selection and customization. It covers embedding model evaluation, fine-tuning embeddings for specific domains, and choosing or training reranker

models. ML engineers and RAG team leads who already have a search system and want to improve its model layer should enter here. This part assumes some familiarity with training neural networks.

Part IV: Evaluation (Chapters 11-13) builds a complete evaluation methodology. It starts with metrics, moves to offline evaluation pipelines with golden test sets and LLM-as-judge approaches, and ends with online experimentation including A/B testing and interleaving experiments for search. Search is one of the few product areas where you can measure quality rigorously, and this part explains how.

Part V: Production Operations (Chapters 14-17) is for the engineers who get paged at 3 AM when search quality degrades and need to know where to look. It covers indexing at scale, latency and throughput optimization, monitoring and observability, and cost optimization. Platform engineers, SREs, and MLOps teams running hybrid search in production should enter here.

Part VI: Applied Domains (Chapters 18-20) applies the preceding material to three specific use cases: RAG pipelines, e-commerce product search, and enterprise knowledge search. These three domains represent the largest deployment categories for hybrid search today, and each imposes constraints that require adapting the general architectural patterns. Each chapter is designed to stand alone as a guide for practitioners in that domain.

The appendices provide a mathematical foundations reference, a guide to benchmark datasets, and a migration playbook for teams moving from legacy keyword search to hybrid architectures.

Table 1.: Reading paths by audience. Ch 4 (reference architecture) is assumed by all subsequent parts. Ch 11 introduces the metrics vocabulary used in Ch 12, Ch 13, and Ch 16. Parts IV and V are independent and can be read in either order.

Audience	Reading Path
Search and platform engineers	Part I (Ch 1–3) → Part II (Ch 4–7) → Part IV (Ch 11–13) → Part V (Ch 14–17) → one domain chapter from Part VI; *optional:* Part III (Ch 8–10)
ML engineers and RAG team leads	Ch 4 → Part III (Ch 8–10) → Part IV (Ch 11–13) → Ch 18
Engineering managers and CTOs	Part I (Ch 1–3) → Ch 7 → Ch 17 → Appendix C; *optional:* Ch 13
Enterprise architects	Part I (Ch 1–3) → Ch 4 + Ch 7 → Ch 14 + Ch 17 → Ch 20

Conventions Used in This Book

Architecture diagrams use a consistent notation throughout. Retrieval stages are shown as sequential pipeline blocks with latency annotations. Where specific vendor APIs are referenced, the examples use the vendor's official Python or JavaScript SDK syntax, but the architectural patterns are vendor-neutral.

When the text references specific metrics, benchmarks, or market data, sources are cited inline. Claims that cannot be attributed to a specific source are framed as general industry observations rather than stated as facts.

A Note on Scope

This book covers the retrieval and ranking components of a search system. It does not cover the full breadth of search-adjacent problems: web crawling and content acquisition pipelines, natural language generation for search summaries, conversational search interfaces, or search-driven recommendation engines. Each of those topics deserves its own treatment.

The book also does not attempt to be a comprehensive survey of every search platform, embedding model, or vector database on the market. The landscape is moving too fast for that. Instead, it provides frameworks for evaluating options so that the guidance remains useful even as specific products evolve.

Where the text references specific tools, versions, and benchmarks, the publication date should be taken into account. The architectural patterns and design principles are intended to be durable. The specific model names and benchmark numbers are not.

Let's build search systems that keep their promises.

Part I.

Why Hybrid Search

1. The Limits of Keyword Search

Every search system makes a promise: give me your question, and I will find the answer. For three decades, the systems keeping that promise have relied on a single core technique: matching the words in a query against the words in an index. This technique, built on term frequency, inverse document frequency, and inverted indexes, has powered every major open-source search engine since the early 1990s. It remains the default today, however it fails, silently and routinely, on a substantial fraction of the queries it receives.

This chapter establishes what keyword search does, why it has dominated for so long, and where it breaks down. The failure modes are specific, measured, and well-documented. Understanding them is the first step toward designing systems that address them.

1.1. The BM25 Foundation

The scoring function at the heart of modern keyword search is BM25, introduced as part of the Okapi information retrieval system in 1994 [8]. The algorithm ranks documents by a weighted sum of term-matching scores. For a query Q containing terms $q_1, q_2, \ldots, q_n$, the score of a document D is:

$$\text{BM25}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{\text{avgdl}}\right)}$$

where $f(q_i, D)$ is the frequency of term q_i in document D, $|D|$ is the document length, avgdl is the average document length in the collection, k_1 controls term frequency saturation (typically 1.2), and b controls document length normalization (typically 0.75). The IDF component is:

$$\text{IDF}(q_i) = \ln\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5} + 1\right)$$

where N is the total number of documents and $n(q_i)$ is the number of documents containing q_i.

The intuition is straightforward. A term that appears frequently in a document but rarely across the collection is a strong relevance signal. The k_1 parameter prevents long documents from dominating simply because they contain more words; term frequency saturates rather than growing linearly. The b parameter penalizes long documents, on the theory that a term appearing once in a 100-word document is more significant than the same term appearing once in a 10,000-word document. These two parameters, along with the IDF weighting that Karen Spärck Jones formalized in 1972 [9], constitute the core of modern lexical ranking.

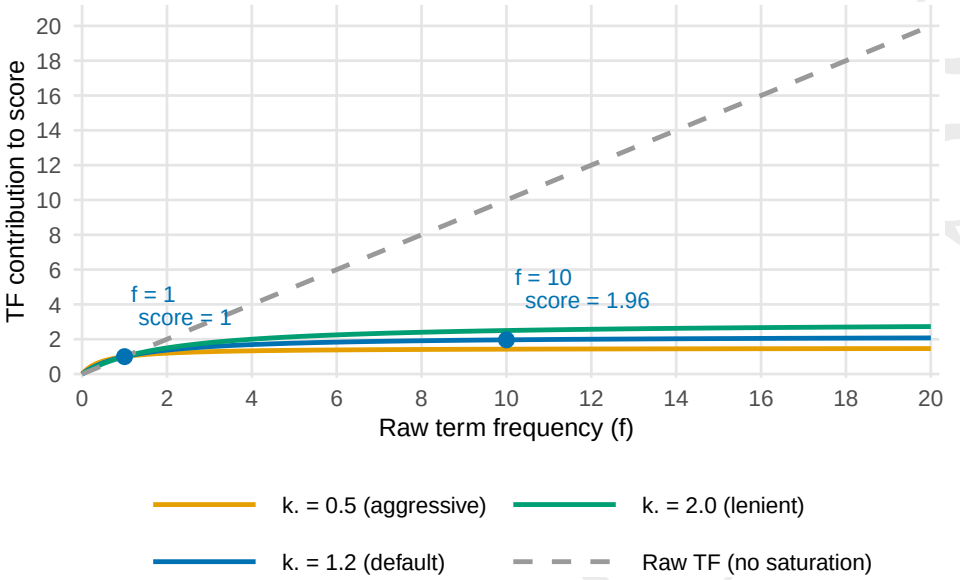

Figure 1.1.: BM25's term frequency component saturates rather than growing linearly. The k_1 parameter controls how quickly. At the default $k_1 = 1.2$, a term's tenth occurrence in a document adds only a fraction of the score contributed by its first occurrence.

The theoretical foundations were formalized comprehensively by the algorithm's creators in a monograph describing BM25 as "one of the most successful text-retrieval algorithms" [10]. That description remains accurate. BM25 is the default ranking function in Elasticsearch (since version 5.0, released in 2016), in OpenSearch, and in Apache Solr (since version 6.0). All three are built on Apache Lucene, which switched its default similarity from TF-IDF to BM25 in Lucene 6. The predecessor TF-IDF framework, rooted in the vector space model [11] and subsequent term-weighting research [12], served as the default for the prior two decades. Between TF-IDF and BM25, term-matching ranking functions have been the production default in the dominant search platforms for over 30 years.

This longevity is not inertia. BM25 has properties that make it

genuinely hard to replace. It requires no training data. It works out of the box on any text collection in any language. It is fast: inverted index lookups with BM25 scoring achieve single-digit millisecond latencies on collections of tens of millions of documents [13]. Its scores are interpretable; an engineer can inspect why a document ranked where it did by examining which query terms matched and how the IDF and term frequency components contributed. For exact-match queries (product SKUs, error codes, model numbers), BM25 is not merely adequate but optimal, because the scoring function directly rewards precise lexical overlap.

These strengths are real and they matter. They are also insufficient.

1.2. The Vocabulary Mismatch Problem

The fundamental limitation of any term-matching system is that it can only find documents that share words with the query. This sounds obvious. The consequences are not.

Two people attempting to describe the same concept will choose the same word less than 20% of the time [14]. This finding, established across five different naming domains (text-editing commands, cooking recipe indexes, classified ad categories, common object descriptions, and information retrieval tasks), is one of the most-cited results in information retrieval. The implication is direct: if a system designer picks one word to label a concept, and a user searches using their own preferred word, the search will fail 80-90% of the time. An earlier, more technical version of the same study reported agreement rates of 10-20% across similar tasks [15].

This is not a failure of lazy users or poorly written documents. It is a structural property of natural language. People use “car” and “automobile” and “vehicle” interchangeably. They search for “cheap flights” when the indexed content says “budget airfare.” They type “laptop

for travel" when the product listing says "ultraportable notebook." In every case, the intent is identical. The words are not.

The problem compounds in specialized domains where multiple vocabularies coexist. Medical professionals search for "myocardial infarction" while patients search for "heart attack." Legal databases contain "tort" and "negligence" while clients search for "personal injury lawsuit." Enterprise knowledge bases are written by specialists and searched by generalists. The vocabulary gap between author and searcher is the norm, not the exception.

A downstream study extended this finding into retrieval settings, showing that an average query term fails to appear in 30-80% of the documents that are actually relevant to the query [16]. In other words, even when a document is exactly what the user needs, there is at least one-in-three chance that it does not contain the search term the user chose. For a multi-term query where all terms must match (the default AND behavior in most search engines), the probability of missing a relevant document grows multiplicatively with each additional term.

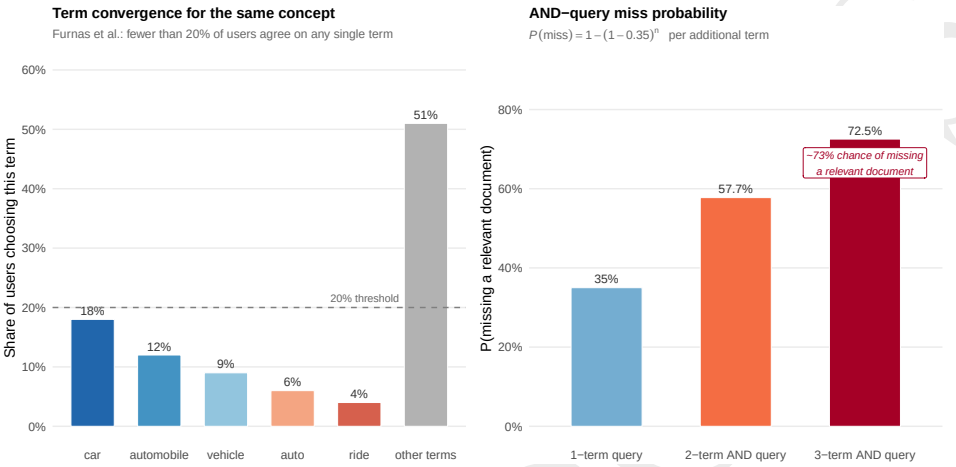

Figure 1.2.: The vocabulary mismatch problem compounds across query terms. Each term has roughly a 35% chance of being absent from a relevant document. For multi-term AND queries, the probability of missing a relevant document grows rapidly.

1.3. The Recall Illusion

The vocabulary mismatch problem would be less dangerous if users could detect it. They cannot.

The most consequential study of retrieval effectiveness in the history of information retrieval evaluated IBM's STAIRS full-text retrieval system on approximately 40,000 documents from litigation related to a San Francisco transit accident [17]. Attorneys and experienced paralegals conducted searches for documents relevant to specific legal issues. They reported that their desired minimum recall was 75% and believed they had achieved at least that level. The actual measured recall was below 20%.

The gap between perceived and actual recall is explained by a simple asymmetry: users see what the system returns, but they do not see what the system misses. When a keyword search returns 50 relevant

documents, the user has no way of knowing that 200 more relevant documents exist in the collection but were not retrieved because they used different terminology. Precision was relatively high in the STAIRS evaluation (mean approximately 79%), which likely reinforced the false confidence. The results looked good. They were just incomplete.

A follow-up evaluation found that when the attorney searched directly instead of delegating to paralegals, mean recall dropped to 15.2% [18]. Adding a thesaurus-linked system found no additional relevant documents beyond those retrieved by plain full-text search, suggesting that vocabulary-based augmentation was insufficient to close the gap.

These results are not merely historical artifacts. The TREC Legal Track, conducted decades later with modern systems, found that keyword searching returned an average of 22% of responsive documents in 2007 [19] and 24% in 2008 [20]. The setting was different (e-discovery rather than litigation research), the technology was newer, and the recall numbers were in the same range.

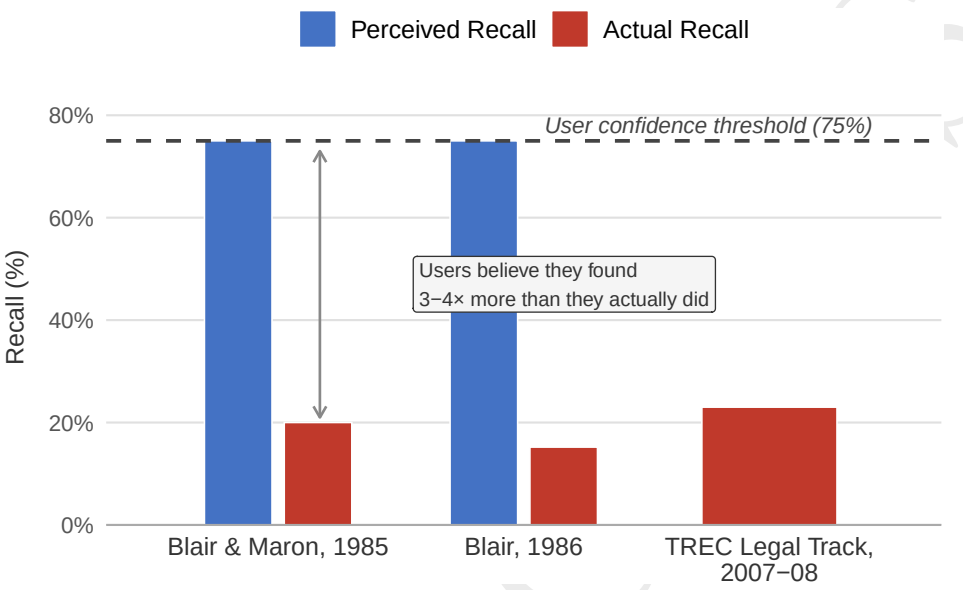

Figure 1.3.: Users consistently overestimate keyword search recall. In the landmark Blair and Maron (1985) study, lawyers believed they had found 75% of relevant documents; actual recall was below 20%. Two decades later, the TREC Legal Track confirmed similar actual recall rates with modern systems.

1.4. Short Queries and Lexical Ambiguity

The vocabulary mismatch problem is amplified by how people actually search. Queries are short. A large-scale analysis of over 51,000 Excite queries in 1997 found a mean query length of 2.21 terms, with roughly one-third of all queries consisting of a single word and four-fifths consisting of three or fewer words [21]. An analysis of approximately one billion AltaVista log entries over a six-week period confirmed similarly short queries and minimal query modification behavior [22]. A comprehensive comparison across nine transaction log studies spanning 1997-2002, representing over one billion queries, found mean query lengths consistently in the range of 2.4 terms [23].

Query lengths have increased modestly over time. Industry data from 2025 puts the average U.S. web query at approximately 3.4 words [24]. This is still short enough to carry substantial ambiguity.

Short queries are more ambiguous than long ones. The average length of queries identified as ambiguous in web search ranges from 1.02 to 1.26 words [25]. An estimated 4% of all web queries and 16% of the most frequent queries are ambiguous in the sense of having multiple plausible interpretations. A query like "jaguar" could refer to the animal, the car brand, or the operating system. "Apple" could mean the fruit or the company. The query "python" could be about the snake, the programming language, or the Monty Python comedy troupe. A lexical search system has no mechanism to disambiguate these: it matches the word, not the meaning.

Even queries that are not classically ambiguous suffer from underspecification. "Best CRM" could mean best for a five-person startup or best for a 10,000-employee enterprise. "Database migration" could refer to moving from MySQL to PostgreSQL, migrating data between schemas, or cloud migration strategy. The problem is not that keyword search returns the wrong results. It is that it returns results based on word overlap rather than intent, and short queries provide very little signal to distinguish among meanings.

The connection between query length and retrieval quality has been formalized through information-theoretic measures. The "clarity score," defined as the relative entropy between the query language model and the collection language model, quantifies how much a query constrains the set of relevant documents [26]. Vague, short queries produce low clarity scores and predict poor retrieval performance. Word sense ambiguity is specifically problematic when retrieving from very short queries [27].

1.5. Traditional Fixes and Their Limits

The information retrieval community has spent decades developing techniques to mitigate vocabulary mismatch within the lexical paradigm. The three most prominent approaches are query expansion, thesaurus-based augmentation (including WordNet), and pseudo-relevance feedback. All three improve average retrieval quality across a set of queries. None of them reliably improves individual queries.

The most comprehensive survey of automatic query expansion techniques states the core problem directly: "current AQE techniques are optimized to perform well on average, but are unstable and may cause degradation of search service for some queries" [28]. This instability is cited as a key reason that automatic query expansion has not been regularly deployed in major commercial search engines.

The most direct form of query expansion adds terms to the original query based on co-occurrence statistics, thesaurus relationships, or user interaction patterns. The risk is topic drift. Expanding "apple" with "fruit, orchard, pie" when the user meant the technology company makes results worse, not better. The expansion that helps one user's interpretation actively harms another's.

Thesaurus-based expansion using resources like WordNet takes a more structured approach, adding synonyms and related terms from a curated lexical database. Testing WordNet synonym sets on TREC collections showed that expansion makes little difference in retrieval effectiveness even when the concepts to be expanded are selected by hand [29]. The emphasis matters: if manually selecting which concepts to expand still fails to improve results reliably, automated selection is unlikely to do better. A later study confirmed that WordNet "may bring many noises for the expansion due to its collection independent characteristic" and cannot keep pace with evolving web vocabulary [30].

Pseudo-relevance feedback (PRF) takes yet another approach: it assumes the top-ranked documents from an initial retrieval are relevant and extracts terms from them to expand the query. This works well when the assumption holds and fails badly when it does not. An analysis of PRF robustness found that "the utilization of QE improves the global performance but hurts the performance on the worst topics" [31]. The need for selective application of query expansion, choosing which queries to expand based on predicted benefit, itself demonstrates that blind application is unreliable. An attempt to find query-level predictors of when expansion would help found no reliable predictors [32].

The instability of these classical techniques persists even with modern models. An evaluation of pseudo-relevance feedback with deep language models and dense retrievers found that on difficult query sets, "all methods hurt MAP and RR" [33]. The fundamental problem is not the technique but the ambiguity of language itself. Statistical and lexical methods can guess at what a user means, but they operate on words rather than meaning, and that distinction becomes critical precisely on the queries where users need the most help.

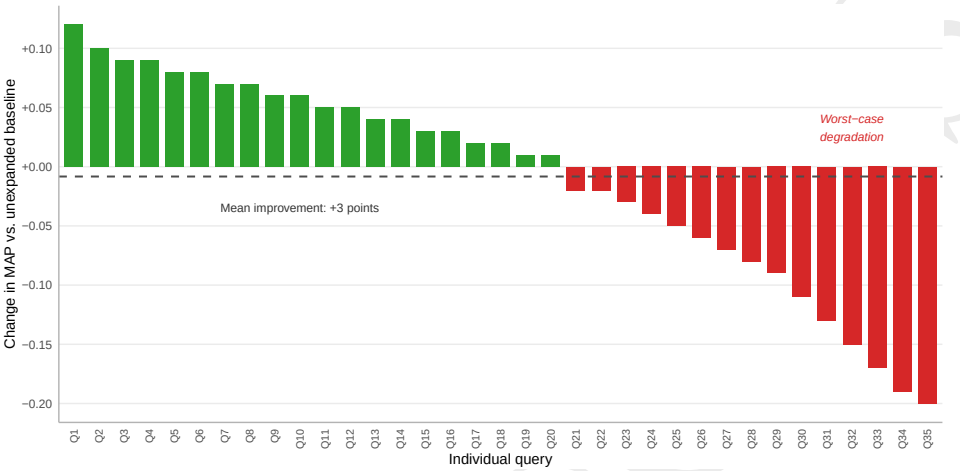

Figure 1.4.: Automatic query expansion improves mean retrieval quality but harms individual queries unpredictably. The positive average masks a long tail of per-query degradation, which is why major search engines have not adopted blind query expansion.

1.6. The Measurable Cost of Lexical Failure

The limitations described above are not theoretical. They have measurable consequences in production systems across industries.

In e-commerce, a benchmark study of the 50 top-grossing U.S. e-commerce sites found that 70% of their search engines were unable to return relevant results for product-type synonyms [34]. When users searched for a synonym of a product type (for example, “notebook” instead of “laptop”), seven out of ten sites failed to return relevant results. The same study found that 29% of all product-finding tasks ended in failure when users relied on search, and 33% of sites could not handle model number searches or single-character misspellings. The failure rate for synonym handling has improved since 2014: a more recent Baymard benchmark found that 41% of sites still fail to fully support the most common search query types [35]. The improvement

is real, but the failure rates remain striking for a technology that has been in production for decades.

In enterprise settings, the cost of search failure translates directly into wasted salary. Multiple studies over two decades have converged on a consistent finding: knowledge workers spend 20-30% of their workday searching for information. A McKinsey Global Institute analysis, drawing on IDC data, estimated that "interaction workers" spend approximately 19% of their time searching and gathering information, with a further 28% managing email [36]. At a 9.5-hour workday (the average for U.S. professionals in the dataset), 19% translates to approximately 1.8 hours per day spent searching. Even at a standard 8-hour day, it is over 90 minutes. IDC estimated that an organization with 1,000 knowledge workers wastes over $5 million annually in salary costs due to time lost searching for information or recreating content that already exists but cannot be found [37], an IDC analyst estimate commissioned by information management vendors. A 2006 analyst report found that information workers spent up to 25% of their day searching for the right information to complete tasks, and estimated that as much as 10% of total salary costs were consumed by inadequate search and discovery [38]. These figures capture more than search engine usage alone; they include browsing, asking colleagues, and navigating file systems. But that breadth is precisely the point: when search does not work, people resort to slower, more expensive methods of finding what they need, and those costs show up as lost productivity whether or not anyone tracks them.

Even at Google's scale, the long tail of language remains unresolved. Approximately 15% of all queries processed by Google every day have never been seen before [39]. This figure has remained remarkably stable: Google reported 25% novel queries in 2007, which dropped to 15% by 2013 and has held steady through at least 2025, confirmed repeatedly by Google engineers at public events [40]. For a system processing billions of queries daily, 15% represents hundreds of millions of queries for which there is no historical click data, no pre-computed expansion, and no prior relevance signal. Each of these queries must

be handled by the retrieval algorithm alone, and for a keyword system, that means pure lexical matching against the index.

1.7. Benchmarking the Gap

The limitations described above are qualitative patterns supported by domain-specific studies. The emergence of large-scale retrieval benchmarks has made it possible to quantify the performance gap between keyword search and neural alternatives across multiple domains.

The BEIR benchmark evaluates retrieval models across 18 diverse datasets spanning biomedical literature, financial documents, question answering, fact checking, and other domains [41]. BM25's average NDCG@10 across the benchmark is approximately 0.412. On the MS MARCO passage ranking dataset, which is the most commonly used training set for neural retrieval models, BM25 underperforms neural first-stage retrievers by 7-18 points in NDCG@10. The gap widens further when a neural reranker is placed on top of BM25's initial retrieval: the BM25 baseline achieves an NDCG@10 of 0.228 on MS MARCO, while a BERT cross-encoder reranking BM25's candidate set reaches 0.401 [41]. That 12-point gain is not a retriever-vs-retriever comparison; it measures how much a neural reranking stage improves BM25's own output. The distinction matters because it demonstrates both BM25's limitations as a standalone ranker and its value as a first-stage candidate generator, a pattern that recurs throughout this book.

Retriever-vs-retriever comparisons tell a more nuanced story. The Dense Passage Retriever (DPR) originally reported gains of 9-19 percentage points in top-20 passage retrieval accuracy over BM25 across four open-domain QA datasets, with the largest gap on Natural Questions [42]. However, a replication study found that the original paper under-reported BM25 baseline effectiveness by approximately 7 points in top-20 accuracy, due to suboptimal parameter tuning [43]. With

properly configured BM25 (tuned k_1 and b parameters, appropriate tokenization), the gap on Natural Questions narrows to roughly 12 points rather than 19. The replication also found that a simple hybrid combination of BM25 and DPR scores outperformed either system alone, a result the original study had not explored. The performance advantage of neural first-stage retrieval over keyword search is real, but the magnitude reported in early benchmarks was inflated by under-tuned lexical baselines. After correction, the retriever-vs-retriever gap on semantic QA tasks is closer to 7-12 points than to 19.

The gap also varies substantially by task type. The largest gains from neural models appear on semantic matching tasks, natural language questions, and queries requiring paraphrase understanding. On tasks that reward exact lexical matching, such as entity retrieval, keyword lookup, and structured queries, BM25 remains highly competitive or superior. This pattern is central to the argument for hybrid search: the failure modes of keyword and neural retrieval are complementary, not overlapping.

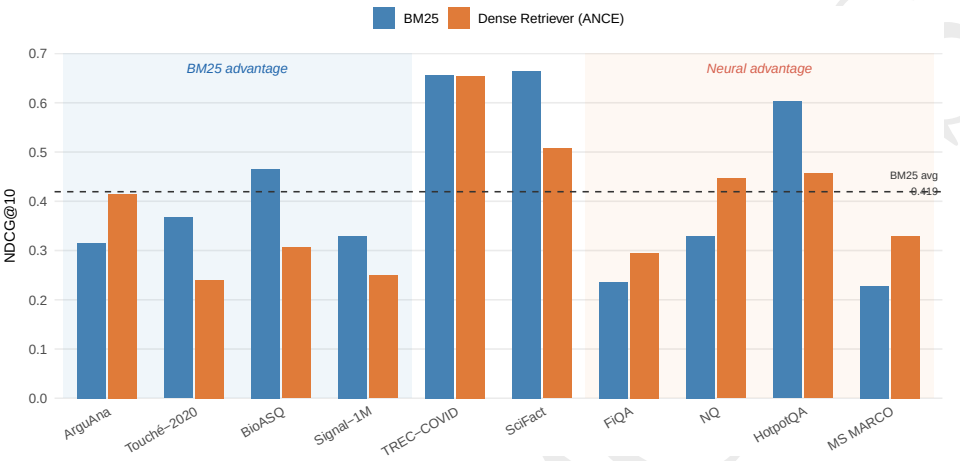

Figure 1.5.: BM25 and neural retrieval excel on different task types. BM25 outperforms dense retrievers on argument retrieval, biomedical queries, and short-text collections, while neural models dominate on semantic QA datasets. Data from the BEIR benchmark.

1.8. What BM25 Does Well

The preceding sections document where keyword search fails. It would be a mistake to conclude that keyword search should be replaced. The evidence for hybrid search depends on understanding not just BM25's weaknesses but its specific, measurable strengths.

Speed is the most obvious advantage. BM25 query latency on the MS MARCO passage collection (approximately 8.8 million passages) is 55 milliseconds, compared to 103 milliseconds for the ANCE dense retriever on the same hardware [44]. With optimized query processing (WAND/Block-Max WAND), BM25 achieves single-digit millisecond p95 latencies for top-1000 retrieval on shards of 100 million documents. The inverted index data structures behind BM25 are compact: a BM25 index for BioASQ's 15 million documents requires approximately 18

GB, compared to roughly 900 GB for ColBERT's full-precision late interaction index on the same corpus [41]. In production systems where latency budgets are strict and infrastructure costs matter, this order-of-magnitude difference in storage and the substantial difference in query latency are significant engineering constraints.

Zero-shot generalization is a less obvious but equally important strength. BM25 requires no training data. It works with default parameters on any text collection in any language. The BEIR benchmark results show that on multiple out-of-domain datasets, BM25 outperforms dense retrieval models that were trained on MS MARCO but evaluated on unfamiliar domains. On Touché-2020, an argument retrieval task, BM25's NDCG@10 of 0.367 remains unbeaten by any neural model tested in the benchmark, including advanced multi-vector systems [41]. On BioASQ, dense retrievers struggle with biomedical terminology (drug names, protein identifiers, gene abbreviations) that embedding models rarely encountered during training. On Signal-1M, a tweet retrieval task, dense retrievers underperform on short texts. Sparse retrieval models appear to achieve better cross-domain generalization than dense retrieval models overall [45].

Interpretability matters for debugging and trust. BM25's scoring function is fully transparent. An engineer can inspect the contribution of each query term to a document's score, identify which terms matched, and understand why a document ranked where it did. When search quality degrades, the diagnosis path is clear: check which terms matched, examine the IDF weights, verify the index contents. Dense retrieval models produce opaque similarity scores from high-dimensional vector comparisons. When a dense retriever returns a confidently wrong result, diagnosing the failure requires analyzing embedding spaces rather than inspecting term matches, a fundamentally harder task.

Exact match handling is BM25's most concrete operational advantage. Product SKUs, error codes, serial numbers, legal case citations, API

endpoint names, and configuration parameters are all cases where the user knows exactly what they want and types it precisely. BM25 handles these queries optimally because they are pure lexical matching tasks. Dense retrieval models, by contrast, may map an exact string to a nearby region of embedding space that contains similar but wrong items. A search for "XPS-13-9340" in a vector space might return results for "XPS-13-9310" or "XPS-15-9530" because the embeddings are close, even though the user wanted an exact model match.

These strengths do not overcome the vocabulary mismatch problem. They do establish that BM25 excels precisely where semantic retrieval struggles. The failure modes are complementary.

Where BM25 Excels:

- Exact match
- Single-digit ms latency, compact index
- Zero-shot (no training data needed)
- Interpretable scoring, debuggable

Where BM25 Fails:

- Synonym matching ("laptop" vs. "notebook")
- Semantic understanding (paraphrase, NL questions)
- Intent disambiguation (ambiguous short queries)
- Negation handling ("without touchscreen")
- Cross-lingual retrieval

BM25's strengths and weaknesses are complementary to those of semantic retrieval. The case for hybrid search rests on this complementarity, not on replacing one approach with the other.

1.9. Summary

Keyword search fails on a substantial fraction of queries due to vocabulary mismatch, a structural property of natural language that

cannot be resolved by matching words alone. Two people choose the same term for the same concept less than 20% of the time [14]. Users cannot detect the resulting recall failures: they believe they have found 75% of what exists when the true figure is closer to 20% [17]. Short queries amplify the problem through lexical ambiguity [21]. Traditional mitigations within the lexical paradigm, including query expansion, thesaurus augmentation, and pseudo-relevance feedback, improve average performance but are unreliable for individual queries [28]. The cost is visible in production: 70% of top e-commerce search engines failing on synonym queries [34], significant portions of the enterprise workday consumed by information seeking [36], and 15% of all Google queries arriving as never-before-seen strings [39]. Neural retrieval benchmarks quantify BM25's underperformance at 7-12 points on semantic QA tasks when baselines are properly tuned, with larger gaps emerging when neural rerankers are layered on top of first-stage retrieval [41], [42], [43], while also revealing that BM25 retains clear advantages in speed, zero-shot generalization, exact match, and interpretability.

The answer, then, is not to replace keyword search. BM25's strengths are too valuable and too difficult to replicate with neural methods. The answer is to augment it. If keyword search fails when users do not use the right words, can the problem be solved by matching meaning instead of words? That is exactly what vector search promises. Chapter 2 examines whether it delivers.

2. The Limits of Vector Search

If keyword search fails when users do not use the right words, matching meaning instead of words should solve the problem. That is exactly what vector search promises. An embedding model encodes a query and a document into the same high-dimensional space, and retrieval becomes a geometric operation: find the document vectors closest to the query vector, regardless of whether they share any words. The promise is real, and for the vocabulary mismatch problem documented in Chapter 1, vector search delivers. But it introduces a new set of failure modes that are, in some ways, harder to detect and harder to fix than the ones it solves. This chapter examines those failure modes systematically.

2.1. What Vector Search Gets Right

The vocabulary mismatch problem is fundamental: two people choose the same term for the same concept with probability less than 0.20 [14]. Keyword search cannot bridge this gap because it operates on token overlap. Dense retrieval operates on learned representations of meaning, which is a fundamentally different approach.

The core idea traces back to the insight that words appearing in similar contexts have similar meanings, and that this similarity can be captured geometrically. Neural networks trained on large text corpora produce word vectors encoding syntactic and semantic regularities, enabling operations like vector(“king”) − vector(“man”) + vector(“woman”)

vector("queen") [46]. Sentence-BERT extended this principle to sentence-level embeddings, producing fixed-length vectors where cosine similarity corresponds to semantic similarity [47]. The practical impact was dramatic: finding the most similar sentence pair among 10,000 sentences dropped from approximately 65 hours (50 million inference computations) with a BERT cross-encoder to roughly 5 seconds with SBERT, while maintaining quality on semantic textual similarity benchmarks.

Dense Passage Retrieval (DPR) applied this approach to information retrieval directly. A dual-encoder trained on question-passage pairs outperformed BM25 by 9 to 19 percentage points in top-20 passage retrieval accuracy on multiple QA datasets. On Natural Questions, DPR achieved 78.4% top-20 accuracy compared to BM25's 59.1% [42]. The qualitative analysis confirmed the mechanism: DPR retrieves correct passages even when there is no lexical overlap between query and passage.

These early results from 2019 and 2020 established the paradigm. The models available today are far more capable. LLM-based embedding models like NV-Embed-v2 (7B parameters), GTE-Qwen2, and E5-Mistral have transformed the aggregate benchmarks. On the 15-dataset BEIR retrieval subset of MTEB, NV-Embed-v2 achieves an average nDCG@10 of 62.65 [48] compared to a BM25 average of approximately 42.3 computed from per-dataset Pyserini baselines over the same 15 datasets [45], a 20-point lead. Where DPR in 2020 beat BM25 on only 1 of 18 BEIR datasets and TAS-B in 2021 won on 8 of 18, NV-Embed-v2 in 2024 wins on 14 of 15. The aggregate gap has not just closed; it has reversed decisively.

A reader encountering this progression might reasonably ask: if modern embeddings dominate BM25 on aggregate benchmarks, do the failure modes documented in the rest of this chapter still matter? They do. The failure surface has migrated from dataset-level averages to per-query analysis. On entity-centric queries, BM25 still outperforms dense embeddings on 40% of individual queries, even when the dataset-

level metric favors the dense model [49]. Product codes, negation, and numerical constraints remain structurally unresolved. The aggregate victory masks specific, systematic failure modes that affect every production search system. The sections that follow document those failure modes with evidence from both the DPR/TAS-B generation and the current generation of embedding models.

Table 2.1.: Keyword retrieval requires token overlap between query and document. Dense retrieval encodes both into a shared vector space, enabling semantic matching at the cost of compressing query meaning into a fixed-length representation.

	Keyword Retrieval (Lexical Match)	Dense Retrieval (Semantic Match)
Input	Query: "lightweight laptop for travel"	Query: "lightweight laptop for travel"
Process	Tokenize → Inverted index lookup	Encode → Query vector [768-dim] → Cosine similarity
"ultraportable notebook"	Not retrieved (no token overlap)	Retrieved (semantically similar vector)
Mechanism	Exact token matching	Learned semantic similarity
Limitation	Misses synonyms and paraphrases	Compresses all query meaning into a fixed-length vector

The problem is that overcoming the lexical gap is not the only thing a search system needs to do.

2.2. Exact Match Degradation

When a user searches for "XPS-13-9340," they need the result for that specific model number, not for the XPS-13-9310 or the XPS-15-9530. Keyword search handles this trivially: the inverted index either contains the token or it does not. Vector search compresses the entire query into a fixed-length dense representation, and in that compression, precise identifiers lose their distinctiveness.

The EntityQuestions dataset tests this failure mode directly. It consists of simple, entity-rich questions such as "Where was Arve Furset born?" On this dataset, DPR drastically underperformed BM25. Performance dropped sharply for rare or uncommon entities, while BM25 remained robust regardless of entity frequency. Data augmentation could not fix the generalization problem; dense retrievers can only handle entities they have seen during training or that closely resemble common patterns [50].

The DPR paper itself acknowledges the limitation. Its qualitative analysis shows cases where BM25 succeeds and DPR fails. For the query "Who plays Thoros of Myr in Game of Thrones?", DPR fails to capture the entity phrase "Thoros of Myr" and retrieves the wrong passage, while BM25 retrieves the correct passage through exact token matching. DPR also underperformed BM25 on the SQuAD dataset overall, an outcome attributed to SQuAD's highly entity-centric nature [42].

These results are from first-generation dense retrievers. Scaling to modern LLM-based embeddings does not resolve the problem. The CapRetrieval benchmark, a fine-grained entity retrieval test, evaluated models from 0.1B to 8B parameters and found that on singleton entity queries, BM25 still outperforms dense embeddings on 40% of individual queries while embeddings win on only 28% [49]. The scale relationship is non-monotonic: GTE-Qwen2 at 1.5B parameters scores 77.35 nDCG@10 on CapRetrieval, worse than GTE-multilingual-base

at 79.67 with roughly 5 times fewer parameters (305M vs 1.5B). E5-Mistral-7B (76.40) underperforms E5-multilingual-large at 0.3B (81.01). Simply making the model larger does not fix entity retrieval.

The underlying problem, which Xu et al. call the **granularity dilemma**, is a structural tension in how embeddings encode information. Training for fine-grained keyword matching causes the encoder to lose grasp of overall semantic importance, and training for semantic matching causes the encoder to lose fine-grained discrimination. Dense encoders "often struggle with fine-grained matching, regardless of training sources or model size" [49]. This is not a solvable optimization problem within the bi-encoder framework. It is an inherent tension between the capacity of a fixed-length vector and the diversity of information it must encode. Sparse models remain "more capable in solving the queries that require exact match (e.g., keyword or entity retrieval), while dense retrievers seem to perform worse in these cases, especially when the entities or compositions are rare or do not appear in the training set" [51].

Table 2.2.: Dense embeddings compress similar identifiers into nearby regions of vector space. An inverted index maintains exact separation. For queries requiring precise identifier matching, this compression is a liability, not a feature.

Identifier	Dense Embedding	Inverted Index
XPS-13-9340	Collapses into "Dell Laptops" region	Exact key → Doc 1
XPS-13-9310	Collapses into "Dell Laptops" region	Exact key → Doc 2
XPS-15-9530	Collapses into "Dell Laptops" region	Exact key → Doc 3
Behavior	Distinct identifiers overlap in vector space	Each identifier is a discrete, unambiguous key

In production, this means any search system that handles product codes, part numbers, error codes, legal case citations, or other precise identifiers will fail on those queries if vector search is the only retrieval path. The scope of the problem is large: 33% of e-commerce sites cannot produce adequate results for exact searches involving product names, model numbers, and SKUs [2]. This existing exact-match problem predates vector search adoption, but dense retrieval does not solve it; if anything, it makes exact-match retrieval less reliable by mapping distinct identifiers into nearby regions of embedding space.

2.3. Entity Confusion and Numerical Blindness

The exact match problem is a special case of a broader limitation: embedding models struggle to separate entities that share surface-level features. The canonical example is "Apple stock price" versus "apple nutrition facts." A contextual embedding model should, in principle, distinguish these because the surrounding words provide context. In practice, the separation is often insufficient for retrieval.

In word embedding spaces, multiple word senses reside in linear superposition. A polysemous word's embedding is approximately a weighted average of its sense embeddings, weighted by the frequency of each sense in the training corpus. The dominant sense dominates the embedding [52]. For "apple," whichever sense appears more frequently in the training data (fruit or company) will pull the embedding in its direction.

Contextualized models (BERT and its descendants) partially address this by conditioning the embedding on surrounding tokens. BERT can place polysemous words into distinct sense regions, but these regions have significant overlap. Word sense disambiguation accuracy ranges from only 59.8% to 81.1% F1 across standard test sets [53]. BERT

struggles more with polysemous pairs (related senses, like "apple" as fruit versus company) than with homonymous pairs (unrelated senses, like "bank" as riverbank versus financial institution). Polysemous word representations do not form isolated clusters but instead form "seamless structures, tightly coupled with sentiment and syntax" [54]. The continuous nature of these representations means that entities sharing a surface form will often remain entangled in embedding space.

The problem is compounded in bi-encoder retrieval architectures, where query and document embeddings are computed independently without cross-attention. A cross-encoder can attend to both the query and document simultaneously, allowing it to disambiguate "apple" based on the full context of the match. A bi-encoder must compress the query "Apple stock price" into a single vector that is already committed to a meaning before it ever sees the document. If the training data is skewed toward one sense of "apple," the embedding will be as well. (Chapter 6 covers the cross-encoder versus bi-encoder distinction in depth, including the latency and quality trade-offs between these architectures.)

Numerical values present a different and arguably more consequential encoding problem, because numerical constraints appear in a large fraction of production queries: price ranges, date filters, version numbers, quantity limits. Unlike polysemy, scaling model size does not help. Qwen3-Embedding at 8B parameters shows erratic cosine similarity between adjacent numbers: the similarity between "49" and "50" may differ substantially from the similarity between "50" and "51," with no consistent monotonic relationship. Surrounding context makes the problem worse, not better; additional text "appears to attenuate fine-grained numerical signals in embeddings" [55].

The retrieval consequence is direct. A query like "hotels under $200" will retrieve documents containing "$400 luxury suites" because the embedding captures the topical similarity (hotels, pricing) while ignoring the numerical constraint. The embeddings for "$200" and "$400"

are not ordered in a way that preserves the "less than" relationship; they are simply two price-related tokens that land in the same region of vector space. The same failure applies to "laptops with at least 16GB RAM," "flights before June 15," or "version 3.2 release notes." In each case, the embedding model captures the topic but discards the numerical precision that defines the user's actual need.

The root cause was identified early: BERT's sub-word tokenization fragments numbers into arbitrary pieces, weakening numerical representations. On the DROP dataset (a numerical reasoning benchmark), the paper's proposed model, NAQANet (a neural QA model combining reading comprehension with simple numerical reasoning), achieved only 32.7% F1, compared to human performance of 96.4% on the same task [56]. Separately, probing studies confirmed that BERT's sub-word tokenization is specifically less effective at encoding numeracy than character-level models, because numbers are split into arbitrary sub-word fragments that destroy their mathematical structure [57]. Modern LLM-based embeddings use different tokenizers but inherit the same fundamental limitation: the embedding space does not preserve numerical ordering or proximity. Queries involving dates, prices, measurements, or version numbers cannot rely on embedding similarity for precise matching.

2.4. Domain-Specific Terminology

Embedding models learn their representations from training data. General-purpose embedding models are trained on broad web corpora, Wikipedia, and common question-answering datasets. When they encounter queries and documents from specialized domains (biomedical literature, legal filings, financial reports, industrial equipment manuals), the representations degrade. The benchmark evidence for this degradation is substantial, and the full dataset-level comparison appears in the "Where BM25 Wins" section later in this chapter. Here,

the focus is on the mechanism: why domain-specific terms fail, and why the failure is difficult to fix.

The explanation is concrete. Terms like "AZD5153" (a specialized biomedical compound identifier) are tokens that the model "most likely has seen only rarely during training. This would cause issues for dense retrievers, as they are unable to represent rare" terms [45]. A keyword search simply matches the token; a dense retriever has no meaningful representation for it. This is not limited to exotic identifiers. General-purpose language models pretrained on broad web text perform poorly on biomedical NLP tasks, with RoBERTa (trained on the largest general pretraining corpus) performing "among the worst" [58]. The word "cold" in general text typically refers to temperature; in medical text, it usually means the common cold. General-purpose embedding models do not capture these domain-specific semantic shifts.

Modern embedding models have narrowed the domain gap on public benchmarks, but often by training on domain-specific data from those benchmarks. NV-Embed-v2 explicitly trains on splits from BioASQ, SciFact, NFCorpus, and other BEIR datasets, so many of its "zero-shot" victories on domain-specific datasets are not truly zero-shot [45]. The implication for practitioners is direct: if surpassing BM25 on domain-specific retrieval requires training on that domain's data, then every new domain a team encounters (their proprietary corpus, their industry's terminology, their customers' vocabulary) presents the original domain gap problem all over again.

Domain adaptation can close the gap, but it is neither free nor simple. Unsupervised domain adaptation via the GPL method improves over out-of-the-box dense retrieval by up to 9.3 nDCG@10 points on specialized datasets, and up to 21.5 points on TREC-COVID when building on TAS-B [59]. But GPL requires generating synthetic training data, running a cross-encoder for pseudo-labeling, and fine-tuning the dense model, all of which require compute, expertise, and iteration per domain. For organizations operating across multiple specialized domains (a common pattern in enterprise search), the cost of domain-adapting

an embedding model for each domain may exceed the cost of simply running BM25 alongside a general-purpose embedding model.

2.5. Hallucinated Similarity

Keyword search has an obvious failure mode: when no documents match the query terms, it returns nothing. The empty results page is a clear signal that something went wrong. Vector search has no equivalent signal. The nearest neighbor to any query vector always exists, regardless of whether the query has any meaningful relationship to the documents in the index. This produces what can be called **hallucinated similarity**: the system returns results with apparent confidence even when no relevant documents exist.

The mathematical foundation for this problem is the **hubness** phenomenon in high-dimensional spaces. As dimensionality increases, the distribution of how often each point appears as a nearest neighbor becomes heavily right-skewed [60]. Certain points become "hubs," appearing as nearest neighbors to many other points regardless of actual semantic relevance. These hubs are an inherent property of high-dimensional data distributions, not a bug in any specific algorithm. In information retrieval, hubness manifests as "obstinate results," documents that persistently appear in retrieval results regardless of the query. These are documents that happen to sit near the centroid of the embedding space, becoming default answers to everything [61].

A related geometric property amplifies the problem. All word representations in BERT, ELMo, and GPT-2 are **anisotropic**, meaning they occupy a narrow cone in embedding space rather than being distributed uniformly [62]. The anisotropic structure means cosine similarity is systematically inflated: two vectors may have high cosine similarity while encoding meaningfully different concepts, simply because both vectors lie in the same narrow region of the space. This makes the similarity scores that vector search returns less discriminative than

they appear, widening the gap between apparent confidence and actual relevance.

Table 2.3.: Keyword search fails visibly (no results) when the query has no lexical overlap with the corpus. Vector search fails invisibly (wrong results with high confidence scores), because the nearest neighbor to any query always exists in the embedding space.

	Keyword Search	Vector Search
Query	"quantum entanglement applications in supply chain management"	"quantum entanglement applications in supply chain management"
Process	Inverted index lookup	Embedding → nearest neighbor search
Result	0 results — "No results found"	3 results returned (similarity 0.71, 0.68, 0.65)
Returned docs	(none)	Quantum computing in cryptography; Quantum key distribution; Post-quantum encryption
Failure mode	Obvious failure — user knows to reformulate	Hallucinated similarity — results look plausible but none answer the query
Key insight	Empty results are a clear signal	The nearest neighbor *always* exists, even when it should not

Distance concentration compounds the problem. Under broadly

applicable conditions, as dimensionality increases, the ratio of the distance to the nearest neighbor and the distance to the farthest neighbor converges to 1 [63]. The practical consequence is that the gap between "close" and "far" shrinks, making the concept of "nearest" less discriminative. The relative contrast between distances to nearest and farthest neighbors diminishes with increasing dimensionality, particularly for L2 (Euclidean) distance [64]. While practical embedding dimensions (768, 1536) are finite and learned representations partially mitigate concentration effects, the general trend holds empirically: similarity scores become less meaningful as a measure of true relevance.

The retrieval-specific consequence is measurable. Dense low-dimensional representations' performance degrades faster than BM25's as index size increases. There exists a tipping point at which sparse representations outperform dense for large enough collections [65]. As the number of documents grows, the probability of false positives (irrelevant documents with high similarity scores) increases, because more documents compete for the same limited embedding space.

For production systems, hallucinated similarity creates a debugging problem that keyword search does not have. When a user searches for "quantum entanglement applications in supply chain management" and the system returns documents about quantum computing in cryptography, the results look plausible. The similarity scores are high. The topics are adjacent. Only careful evaluation reveals that the results do not answer the actual query. At scale, this failure mode is invisible without explicit relevance measurement (a topic Part IV covers in detail).

2.6. Negation, Numerical Constraints, and Boolean Logic

Embedding similarity is fundamentally a measure of topical relatedness, not logical entailment. This distinction becomes visible on

queries involving negation, numerical constraints, or Boolean conditions. "Laptops with touchscreen" and "laptops without touchscreen" are about the same topic (laptops, touchscreens) but express opposite requirements. Embedding models treat them as nearly identical.

The evidence for this failure is extensive and consistent across model families. Pretrained language models do not distinguish between negated and non-negated cloze statements; "Birds cannot [MASK]" and "Birds can [MASK]" produce essentially the same predictions [66]. On the NEG-136 benchmark, for negative sentences like "A robin is not a _____," BERT prefers the true completion in 0% of items, assigning higher probability to the false completion (e.g., "bird") every single time. The completions for "A robin is a [MASK]" and "A robin is not a [MASK]" are functionally identical [67].

Since embedding models are derived from these same transformer architectures, the insensitivity to negation transfers directly to the embedding space. State-of-the-art text embedding models (GTE, BGE, E5, Gecko, LLM2Vec) mostly fail a simple negation test: the cosine similarity between an original sentence and its negation is nearly as high as between the original and a genuine paraphrase. The underlying conflict is between "topic information" and "negation information" in the embedding space; embeddings prioritize topic similarity over logical semantics [68]. The MTEB benchmark, the standard evaluation for embedding models, does not adequately test negation awareness.

Practitioner testing confirms the scale of the problem in production settings. Similarity scores between "The treatment improved patient outcomes" and "The treatment did not improve patient outcomes" have been measured at 0.96 cosine similarity. "Home with pool" versus "home without pool" produced 0.82 cosine similarity, compared to 0.13 for truly unrelated queries [69]. Negated and non-negated sentences are consistently placed "very close to each other in the representation space with high similarity" [70].

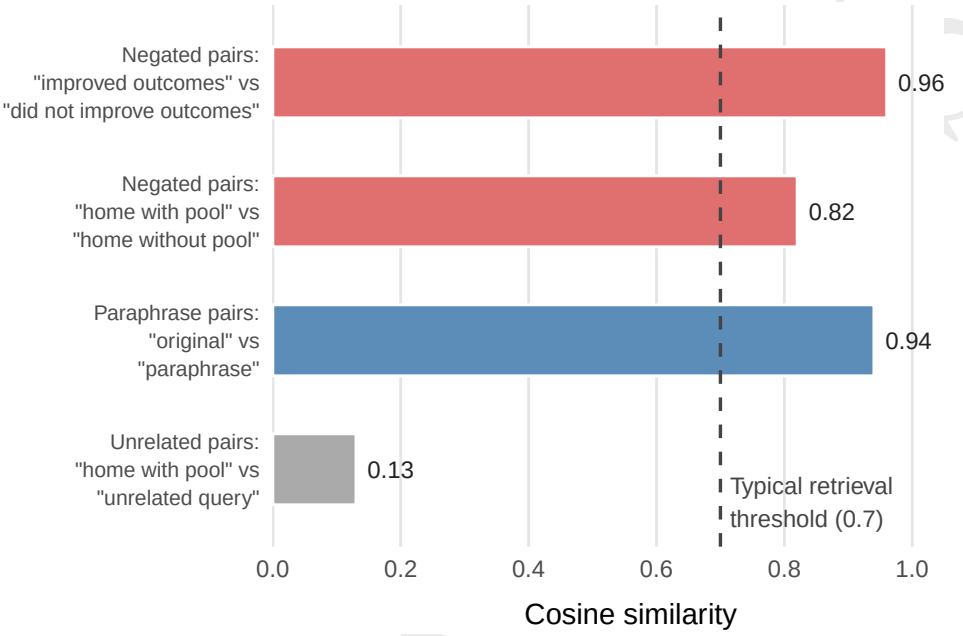

Figure 2.1.: Cosine similarity between sentences and their negations is nearly indistinguishable from similarity between sentences and their paraphrases. A negated medical statement (0.96) would rank above many semantically equivalent reformulations.

The root cause is in the training data. Only 1.19% of SNLI pairs and 7.16% of RTE pairs contain negation. Of those, nearly half are unimportant: one can ignore the negation and still make the correct inference [71]. Models trained on this data have no incentive to learn negation sensitivity. Across eight popular NLU corpora, few negations appear compared to general English, and state-of-the-art transformers obtain "substantially worse results with instances that contain negation, especially if the negations are important" [72].

The implication for search is direct. Any query that includes "without," "not," "excluding," "other than," or any numerical constraint (under $500, before 2020, at least 16GB) cannot be reliably handled by embedding similarity alone. These queries require either a

keyword-based filtering mechanism, structured metadata filtering, or a reranking model that evaluates query-document pairs jointly rather than independently.

2.7. Production Complexity

Even setting aside all quality limitations, the "just use embeddings" approach introduces operational complexity that keyword search does not have. The structural cost relationships are worth understanding independently of any specific price point, because they constrain every vector search deployment regardless of which vendor or hardware generation is involved.

The first structural cost is that every document must be encoded by the embedding model, and every query must be encoded at search time. This per-document, per-query compute requirement has no equivalent in keyword search, where indexing is a string operation and query execution is a lookup. As of early 2026, a single NVIDIA L4 GPU processes approximately 2,000 text tokens per second through a 7B parameter embedding model. Generating embeddings for a billion-item collection requires over 5.8 days on a single GPU [73]. These specific throughput numbers will improve with new hardware, but the structural requirement of running a neural network forward pass per document remains.

The second structural cost is storage. Vector index storage scales as $O(n \times d)$, where n is the number of documents and d is the embedding dimensionality. Inverted index storage scales with vocabulary size and document count, but vocabulary grows sublinearly with corpus size. The concrete difference is substantial: on the DBPedia corpus (approximately 1 million documents), BM25 requires 0.4 GB of storage; dense retrieval at 768 dimensions requires 3 GB; ColBERT (128-dimension multi-vector representation) requires 20 GB. At 15 million documents, ColBERT requires approximately 900 GB compared to

BM25's 18 GB, a 50-fold overhead [41]. The absolute numbers change with quantization and compression (Chapter 17 covers these techniques in detail), but the ratio between sparse and dense storage remains significant at any scale.

Approximate nearest neighbor (ANN) search trades recall for speed, adding another source of error. Production vector databases use algorithms like HNSW, which builds a navigable graph structure enabling logarithmic-time approximate search [74]. The approximation is controlled by the `ef` parameter (the beam width during search). On the SIFT-1M benchmark with M=16, the hnswlib reference implementation achieves 0.713 recall at ef=10, 0.950 at ef=50, and 0.985 at ef=100, reaching near-perfect recall only above ef=500 at the cost of a 17-fold reduction in queries per second (ANN-Benchmarks, SIFT-1M results, hnswlib with M=16, efConstruction=500). These are recall figures for finding true nearest neighbors, not for search relevance, but they compound with all the quality issues already discussed. If the true nearest neighbor is already only somewhat relevant (due to entity confusion, negation blindness, or domain shift), returning an approximate nearest neighbor that is slightly farther away makes the result even less relevant.

The memory requirements are substantial. HNSW requires the full graph to reside in RAM for fast access. The arithmetic is straightforward: 100 million vectors at 768 dimensions in float32 require 100M x 768 x 4 bytes = 307 GB for raw vector storage alone. The HNSW graph adds connection metadata; at M=16 (a common default), each node maintains up to 32 bidirectional edges at the base layer, adding approximately 100M x 32 x 4 bytes = 12.8 GB of graph structure. With indexing overhead, a 100-million-vector HNSW index at 768 dimensions requires roughly 350 GB of RAM; at 1536 dimensions (common for OpenAI and Cohere embeddings), the figure exceeds 600 GB. At billion scale, memory costs become prohibitive without disk-based alternatives like DiskANN, which reduce RAM requirements by 15 to 50 times but introduce additional latency [75].

Filtering introduces a particularly painful trade-off. Production search almost always requires combining similarity with exact constraints: in-stock items only, documents the user has permission to access, products in a specific price range. HNSW was not designed for filtered search. Pre-filtering (restricting the candidate set before ANN search) disrupts graph connectivity, causing fragmented search paths and lower accuracy. Post-filtering (running ANN search first, then removing non-matching results) can discard too many results, especially with low-cardinality filters [76]. Pre-filtering with HNSW increases distance computation latency by 128% compared to unfiltered search, while post-filtering increases latency by 75% [77]. Across the industry, hybrid queries combining attributes and vectors remain a challenge; "the cost of rebalancing vectors across different segments is very high due to the data volume and computing complexity" [78].

Notion's production experience illustrates how these structural costs compound. After launching vector search in 2024, their initial vector database ran out of capacity within one month. Scaling required an 8-fold increase in database capacity, and even after a migration that achieved 50% cost reduction, the annual run rate for vector database costs remained in the millions of dollars. Embedding model upgrades required full re-indexing of the entire corpus, and the team had to build a dual-path architecture (batch plus real-time) for indexing to maintain freshness [79].

None of this means vector search is not worth the investment. It means that vector search is not a simple replacement for keyword search. It is an additional system with its own infrastructure requirements, failure modes, and operational costs.

2.8. Where BM25 Wins: The Benchmark Evidence

The narrative around dense retrieval often emphasizes aggregate benchmark scores, and by that measure, modern embedding models have won decisively. NV-Embed-v2 outperforms BM25 on 14 of 15 BEIR datasets in the MTEB evaluation, with an average nDCG@10 lead of approximately 20 points [48]. The era in which BM25 dominated the majority of BEIR datasets is over.

In 2021, TAS-B, the best single-vector dense model in the original BEIR benchmark, lost to BM25 on 10 of 18 datasets. The margins were large: 20.5 nDCG@10 points on Touché-2020, 17.5 points on TREC-COVID, 8.2 points on BioASQ [41]. DPR performed 47.7% worse than BM25 on average across all 18 BEIR datasets. By 2024, NV-Embed-v2 had flipped the ledger, losing on only 1 of 15 datasets. The domain-specific reversals are dramatic: on TREC-COVID, NV-Embed-v2 scores 0.884 compared to BM25's 0.656; on SciFact, 0.801 versus 0.665 [48]. But as discussed in the domain-specific terminology section, these gains come partly from training on domain data, raising questions about true zero-shot generalization [45]. The gap has closed not because the failure modes disappeared, but because modern models are large enough and trained on enough data to compensate on aggregate while still failing on specific query types.

Those specific failures are visible only when evaluation moves from dataset-level metrics to per-query analysis, which is where users actually experience search quality. The most revealing analysis comes from the CapRetrieval benchmark. Even when the dataset-level nDCG@10 favors the dense model, BM25 outperforms dense retrieval on 40% of individual entity-centric queries [49]. The error taxonomy from this benchmark maps directly to production failure modes: "literal errors" where passages containing exact query terms are ranked low (BM25 would succeed trivially), "over-generalization" where "shoe" retrieves "cardboard box with anti-trample symbol," and "condition ignoring"

where "purple flower" retrieves "purple butterflies" because the model captures the attribute but not the entity.

The one BEIR dataset where BM25 has retained its advantage across four years and every generation of neural models is Touché-2020, an argument retrieval task. BM25's nDCG@10 of 0.367 on this dataset remains unbeaten by all neural models tested, including multi-vector systems [80]. The cause is a structural mismatch between neural encoding and argument quality: neural models produce fixed-size embeddings regardless of document length, which biases them toward short, high-overlap passages. Substantive long arguments, which are what the task requires, are systematically underranked. Even after denoising the dataset to remove problematic short documents, BM25 still wins.

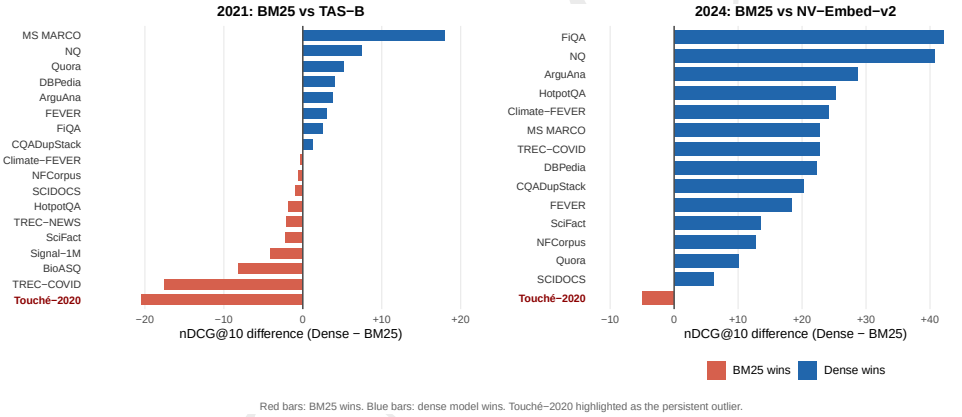

Figure 2.2.: The BM25 vs. dense retrieval landscape has transformed between 2021 and 2024. TAS-B lost to BM25 on 10 of 18 BEIR datasets; NV-Embed-v2 loses on only 1 of 15. But dataset-level victories mask per-query failures on entity-centric, exact-match, and argument retrieval queries.

There is a theoretical explanation for why per-query failures persist even as aggregate metrics improve. Fixed-length encodings face inherent "limitations in the capacity of fixed-length encodings to support

precise retrieval of long documents" [81]. Sparse representations scale naturally with vocabulary; dense representations are constrained by their fixed dimensionality. Only two retrieval architectures consistently outperform BM25 on average across the full BEIR benchmark without training on BEIR data: ColBERT-style multi-vector models, and BM25 combined with a cross-encoder reranker. Both are, architecturally, hybrid approaches.

This does not mean vector search should be abandoned. It means that vector search should not be the sole retrieval mechanism. The engineering decision is not "keyword or vector" but "how to combine them."

2.9. Summary

Chapter 1 showed where keywords fail. This chapter showed where vectors fail. The failure modes are complementary: keyword search excels at exact match, entity retrieval, and structured filtering, precisely where vector search struggles. Vector search handles semantic matching, synonymy resolution, and natural language queries, the categories where keyword search has no answer. Even with modern embedding models dominating aggregate benchmarks, BM25 still outperforms dense retrieval on 40% of individual entity-centric queries [49]. The fact that modern models have closed the aggregate gap makes the case for hybrid search more precise, not less relevant. The failures have migrated from being visible in dataset-level averages to being visible only in per-query analysis, which makes them harder to detect but no less consequential for production systems where every missed product code or mishandled negation is a user-facing error. Chapter 3 examines how to combine the two approaches.

References

[1] The Harris Poll, "Retail search consumer survey," 2021. Available: https://cloud.google.com/blog/topics/retail/search-abandonment-impacts-retail-sales-brand-loyalty

[2] Baymard Institute, "Deconstructing e-commerce search UX: The 8 most common search query types (41." 2024. Available: https://baymard.com/blog/ecommerce-search-query-types

[3] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to information retrieval*. Cambridge University Press, 2008.

[4] Next Move Strategy Consulting, "Retrieval-augmented generation (RAG) market report," 2025. Available: https://www.nextmsc.com/report/retrieval-augmented-generation-rag-market-ic3918

[5] Global Market Insights, "Vector database market size & share 2026-2034." 2024. Available: https://www.gminsights.com/industry-analysis/vector-database-market

[6] V. Vaibhav and M. Ludack, "From query to cart: Inside Target's search bar overhaul with AlloyDB AI." 2025. Available: https://cloud.google.com/blog/topics/retail/from-query-to-cart-inside-targets-search-bar-overhaul-with-alloydb-ai

[7] D. Haney and D. Gibson, "Ask like a human: Implementing semantic search on Stack Overflow." [Online]. Available: https://stackoverflow.blog/2023/07/31/ask-like-a-human-implementing-semantic-search-on-stack-overflow/

[8] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford, "Okapi at TREC-3," in *Proceedings of the third text REtrieval conference (TREC-3)*, in NIST special publication 500-225. 1994, pp. 109–126.

[9] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of Documentation*, vol. 28, no. 1, pp. 11–21, 1972, doi: 10.1108/eb026526.

[10] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: BM25 and beyond," *Foundations and Trends in Information Retrieval*, vol. 3, no. 4, pp. 333–389, 2009, doi: 10.1561/1500000019.

[11] G. Salton, A. Wong, and C.-S. Yang, "A vector space model for automatic indexing," *Communications of the ACM*, vol. 18, no. 11, pp. 613–620, 1975, doi: 10.1145/361219.361220.

[12] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988.

[13] A. Grand, "Why BM25 queries with more terms can be faster (and other scaling surprises)." 2026. Available: https://turbopuffer.com/blog/bm25-latency-musings

[14] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais, "The vocabulary problem in human-system communication," *Communications of the ACM*, vol. 30, no. 11, pp. 964–971, 1987, doi: 10.1145/32206.32212.

[15] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais, "Statistical semantics: Analysis of the potential performance of key-word information systems," *Bell System Technical Journal*, vol. 62, no. 6, pp. 1753–1806, 1983.

[16] L. Zhao and J. Callan, "Term necessity prediction," in *Proceedings of CIKM '10*, 2010.

[17] D. C. Blair and M. E. Maron, "An evaluation of retrieval effectiveness for a full-text document-retrieval system," *Communications of the ACM*, vol. 28, no. 3, pp. 289–299, 1985, doi: 10.1145/3166.3197.

[18] D. C. Blair, "Full text retrieval: Evaluation and implications," *International Classification*, vol. 13, no. 1, pp. 18–23, 1986.

[19] S. Tomlinson, D. W. Oard, J. R. Baron, and P. Thompson, "Overview of the TREC 2007 legal track," in *Proceedings of the sixteenth text REtrieval conference (TREC 2007)*, NIST, 2007. Available: https://trec.nist.gov/pubs/trec16/papers/LEGAL.OVERVIEW16.pdf

[20] D. W. Oard, B. Hedin, S. Tomlinson, and J. R. Baron, "Overview of the TREC 2008 legal track," in *Proceedings of the seventeenth text REtrieval conference (TREC 2008)*, NIST, 2008. Available: https://trec.nist.gov/pubs/trec17/papers/LEGAL.OVERVIEW08.pdf

[21] B. J. Jansen, A. Spink, and T. Saracevic, "Real life, real users, and real needs: A study and analysis of user queries on the web," *Information Processing & Management*, vol. 36, no. 2, pp. 207–227, 2000, doi: 10.1016/S0306-4573(99)00056-4.

[22] C. Silverstein, M. Henzinger, H. Marais, and M. Moricz, "Analysis of a very large web search engine query log," *ACM SIGIR Forum*, vol. 33, no. 1, pp. 6–12, 1999, doi: 10.1145/331403.331405.

[23] B. J. Jansen and A. Spink, "How are we searching the world wide web? A comparison of nine search engine transaction logs," *Information Processing and Management*, vol. 42, no. 1, pp. 248–263, 2006.

[24] Semrush, "29 eye-opening Google search statistics for 2025." 2025. Available: https://www.semrush.com/blog/google-search-statistics/

[25] M. Sanderson, "Ambiguous queries: Test collections need more sense," in *Proceedings of SIGIR '08*, 2008, pp. 499–506. doi: 10.1145/1390334.1390420.

[26] S. Cronen-Townsend and W. B. Croft, "Quantifying query ambiguity," in *Proceedings of HLT 2002*, 2002, pp. 94–98.

[27] R. Krovetz and W. B. Croft, "Lexical ambiguity and information retrieval," *ACM Transactions on Information Systems*, vol. 10, no. 2, pp. 115–141, 1992.

[28] C. Carpineto and G. Romano, "A survey of automatic query expansion in information retrieval," *ACM Computing Surveys*, vol. 44, no. 1, pp. 1:1–1:50, 2012, doi: 10.1145/2071389.2071390.

[29] E. M. Voorhees, "Query expansion using lexical-semantic relations," in *Proceedings of SIGIR '94*, 1994, pp. 61–69. doi: 10.1007/978-1-4471-2099-5_7.

[30] Z. Gong, C. W. Cheang, and L. H. U, "Web query expansion by WordNet," in *Database and expert systems applications (DEXA 2005)*, in LNCS, vol. 3588. 2005, pp. 166–175. doi: 10.1007/11546924_17.

[31] G. Amati, C. Carpineto, and G. Romano, "Query difficulty, robustness, and selective application of query expansion," in *Advances in information retrieval (ECIR 2004)*, in LNCS, vol. 2997. 2004, pp. 127–137. doi: 10.1007/978-3-540-24752-4_10.

[32] B. Billerbeck and J. Zobel, "When query expansion fails," in *Proceedings of SIGIR '03*, 2003, pp. 387–388. doi: 10.1145/860435.860503.

[33] H. Li, A. Mourad, S. Zhuang, B. Koopman, and G. Zuccon, "Pseudo relevance feedback with deep language models and dense retrievers: Successes and pitfalls," *ACM Transactions on Information Systems*, 2023, doi: 10.1145/3570724.

[34] Baymard Institute, "E-commerce search usability: Report & benchmark." 2014. Available: https://baymard.com/blog/ecommerce-search-report-and-benchmark

[35] Baymard Institute, "Deconstructing e-commerce search UX: The 8 most common search query types." 2024. Available: https://baymard.com/blog/ecommerce-search-query-types

[36] McKinsey Global Institute, "The social economy: Unlocking value and productivity through social technologies," Jul. 2012.

[37] S. Feldman and C. Sherman, "The high cost of not finding information," IDC, 29127, Apr. 2003.

[38] Butler Group, "Enterprise search and retrieval," 2006.

[39] P. Nayak, "Understanding searches better than ever before." Oct. 2019. Available: https://blog.google/products/search/search-language-understanding-bert/

[40] J. Mueller, "Remarks at search central live NYC." 2025.

[41] N. Thakur, N. Reimers, A. Rücklé, A. Srivastava, and I. Gurevych, "BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models," in *NeurIPS 2021 datasets and benchmarks track*, 2021. Available: https://arxiv.org/abs/2104.08663

[42] V. Karpukhin *et al.*, "Dense passage retrieval for open-domain question answering," in *Proceedings of EMNLP 2020*, 2020, pp. 6769–6781. doi: 10.18653/v1/2020.emnlp-main.550.

[43] X. Ma, K. Sun, R. Pradeep, M. Li, and J. Lin, "Another look at DPR: Reproduction of training and replication of retrieval," in *ECIR 2022*, 2022, pp. 613–626.

[44] J. Mackenzie, A. Trotman, and J. Lin, "Predicting efficiency/effectiveness trade-offs for dense vs. Sparse retrieval strategy selection," 2021, Available: https://arxiv.org/abs/2109.10739

[45] E. Kamalloo, X. C. Zhang, *et al.*, "Resources for brewing BEIR: Reproducible reference models and an official leaderboard," in *Proceedings of SIGIR 2024*, 2024.

[46] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st international conference on learning representations (ICLR 2013), workshop track*, 2013. Available: https://arxiv.org/abs/1301.3781

[47] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using siamese BERT-networks," in *Proceedings of EMNLP-IJCNLP 2019*, 2019, pp. 3982–3992. doi: 10.18653/v1/D19-1410.

[48] M. Lee, N. Jain, D. Cer, and Y. Hou, "NV-Embed: Improved techniques for training LLMs as generalist embedding models," in *Proceedings of ICLR 2025*, 2025.

[49] L. Xu, Z. Su, M. Yu, J. Li, F. Meng, and J. Zhou, "Dense retrievers can fail on simple queries: Revealing the granularity dilemma of embeddings," in *Findings of EMNLP 2025*, 2025, pp. 19295–19305.

[50] C. Sciavolino, Z. Zhong, J. Lee, and D. Chen, "Simple entity-centric questions challenge dense retrievers," in *Proceedings of EMNLP 2021*, 2021, pp. 6138–6148. doi: 10.18653/v1/2021.emnlp-main.496.

[51] W. X. Zhao *et al.*, "Dense text retrieval based on pretrained language models: A survey," *ACM Transactions on Information Systems*, 2024, doi: 10.1145/3637870.

[52] S. Arora, Y. Li, Y. Liang, T. Ma, and A. Risteski, "Linear algebraic structure of word senses, with applications to polysemy," *Transactions of the Association for Computational Linguistics*, vol. 6, pp. 483–495, 2018.

[53] G. Wiedemann, S. Remus, A. Chawla, and C. Biemann, "Does BERT make any sense? Interpretable word sense disambiguation with contextualized embeddings," 2019, Available: https://arxiv.org/abs/1909.10430

[54] D. Yenicelik, F. Schmidt, and Y. Kilcher, "How does BERT capture semantics? A closer look at polysemous words," in *Proceedings of the third BlackboxNLP workshop on analyzing and interpreting neural networks for NLP*, 2020, pp. 156–162.

[55] M. M. Borkakoty, P. Nakov, and M. Hakami, "Revealing the numeracy gap: An empirical investigation of text embedding models," 2025, Available: https://arxiv.org/abs/2509.05691

[56] D. Dua, Y. Wang, P. Dasigi, G. Stanovsky, S. Singh, and M. Gardner, "DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs," in *Proceedings of NAACL-HLT 2019*, 2019, pp. 2368–2378. doi: 10.18653/v1/N19-1246.

[57] E. Wallace, Y. Wang, S. Li, S. Singh, and M. Gardner, "Do NLP models know numbers? Probing numeracy in embeddings," in *Proceedings of EMNLP-IJCNLP 2019*, 2019, pp. 5307–5315. doi: 10.18653/v1/D19-1534.

[58] Y. Gu *et al.*, "Domain-specific language model pretraining for biomedical natural language processing," *ACM Transactions on Computing for Healthcare*, vol. 3, no. 1, pp. 1–23, 2022, doi: 10.1145/3458754.

[59] K. Wang, N. Thakur, N. Reimers, and I. Gurevych, "GPL: Generative pseudo labeling for unsupervised domain adaptation of dense retrieval," in *Proceedings of NAACL 2022*, 2022, pp. 2345–2360. doi: 10.18653/v1/2022.naacl-main.168.

[60] M. Radovanović, A. Nanopoulos, and M. Ivanović, "Hubs in space: Popular nearest neighbors in high-dimensional data," *Journal of Machine Learning Research*, vol. 11, pp. 2487–2531, 2010.

[61] M. Radovanović, A. Nanopoulos, and M. Ivanović, "On the existence of obstinate results in vector space models," in *Proceedings of the 33rd annual international ACM SIGIR conference on research and development in information retrieval*, 2010, pp. 186–193.

[62] K. Ethayarajh, "How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings," in *Proceedings of EMNLP-IJCNLP 2019*, 2019, pp. 55–65. doi: 10.18653/v1/D19-1006.

[63] K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is nearest neighbor meaningful?" in *Proceedings of the 7th international conference on database theory (ICDT)*, in LNCS, vol. 1540. 1999, pp. 217–235.

[64] C. C. Aggarwal, A. Hinneburg, and D. A. Keim, "On the surprising behavior of distance metrics in high dimensional space," in *Proceedings of the 8th international conference on database theory (ICDT)*, in LNCS, vol. 1973. 2001, pp. 420–434.

[65] N. Reimers and I. Gurevych, "The curse of dense low-dimensional information retrieval for large index sizes," in *Proceedings of ACL-IJCNLP 2021 (volume 2: Short papers)*, 2021, pp. 605–611. doi: 10.18653/v1/2021.acl-short.77.

[66] N. Kassner and H. Schütze, "Negated and misprimed probes for pretrained language models: Birds can talk, but cannot fly," in *Proceedings of ACL 2020*, 2020, pp. 7811–7818. doi: 10.18653/v1/2020.acl-main.698.

[67] A. Ettinger, "What BERT is not: Lessons from a new suite of psycholinguistic diagnostics for language models," *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 34–48, 2020, doi: 10.1162/tacl_a_00298.

[68] H. Cao, "Enhancing negation awareness in universal text embeddings: A data-efficient and computational-efficient approach," 2025, Available: https://arxiv.org/abs/2504.00584

[69] J. Sack, "The negation problem in vector search." Dec. 2025.

[70] M. Rohit, L. Jeganathan, S. R. Ummity, M. J. Meena, and J. Balabaskaran, "Computation of sentence similarity score through hybrid deep learning with a special focus on negation sentence," *Scientific Reports*, 2026, doi: 10.1038/s41598-025-34084-2.

[71] M. M. Hossain, V. Kovatchev, P. Dutta, T. Kao, E. Wei, and E. Blanco, "An analysis of natural language inference benchmarks through the lens of negation," in *Proceedings of EMNLP 2020*, 2020, pp. 9106–9118. doi: 10.18653/v1/2020.emnlp-main.732.

[72] M. M. Hossain, D. Chinnappa, and E. Blanco, "An analysis of negation in natural language understanding corpora," in *Proceedings of ACL 2022 (volume 2: Short papers)*, 2022, pp. 716–723. doi: 10.18653/v1/2022.acl-short.81.

[73] Introl, "Embedding infrastructure at scale: Vector generation for production AI." Dec. 2025. Available: https://introl.com/blog/embedding-infrastructure-scale-vector-generation-production-guide-2025

[74] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 4, pp. 824–836, 2020, doi: 10.1109/TPAMI.2018.2889473.

[75] S. J. Subramanya, F. Devvrit, H. V. Simhadri, R. Krishnaswamy, and R. Kadekodi, "DiskANN: Fast accurate billion-point nearest neighbor search on a single node," in *Advances in neural information processing systems (NeurIPS)*, 2019, pp. 13748–13758.

[76] Y. Chronis *et al.*, "Filtered vector search: State-of-the-art and research opportunities," *Proceedings of the VLDB Endowment*, vol. 18, no. 12, pp. 5488–5492, 2025.

[77] Filter-Centric Vector Indexing, "Filter-centric vector indexing: Geometric transformation for efficient filtered vector search," in *aiDM '25*, 2025.

[78] J. Wang *et al.*, "Milvus: A purpose-built vector data management system," in *Proceedings of SIGMOD '21*, 2021, pp. 2614–2627.

[79] P. Gondi, M. Liu, N. Louie, C. Lund, and J. Sager, "Two years of vector search at Notion: 10x scale, 1/10th cost." Feb. 2026. Available: https://www.notion.com/blog/two-years-of-vector-search-at-notion

[80] N. Thakur, E. Kamalloo, T. Formal, C. Lassance, R. Pires, and J. Lin, "Systematic evaluation of neural retrieval models on the touché 2020 argument retrieval subset of BEIR," 2024, Available: https://arxiv.org/abs/2407.07790

[81] Y. Luan, J. Eisenstein, K. Toutanova, and M. Collins, "Sparse, dense, and attentional representations for text retrieval," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 329–345, 2021, doi: 10.1162/tacl_a_00369.

[82] D. Lee, S. Hwang, K. Lee, S. Choi, and S. Park, "On complementarity objectives for hybrid retrieval," in *Proceedings of the 61st annual meeting of the association for computational linguistics (ACL 2023)*, 2023, pp. 13357–13368.

[83] X. Chen *et al.*, "Salient phrase aware dense retrieval: Can a dense retriever imitate a sparse one?" in *Findings of the association for computational linguistics: EMNLP 2022*, 2022, pp. 250–262. doi: 10.18653/v1/2022.findings-emnlp.19.

[84] G. V. Cormack, C. L. A. Clarke, and S. Büttcher, "Reciprocal rank fusion outperforms condorcet and individual rank learning methods," in *Proceedings of the 32nd international ACM SIGIR conference (SIGIR '09)*, 2009, pp. 758–759. doi: 10.1145/1571941.1572114.

[85] R. Benham and J. S. Culpepper, "Risk-reward trade-offs in rank fusion," in *Proceedings of the 22nd australasian document computing symposium (ADCS '17)*, 2017, pp. 1–8. doi: 10.1145/3166072.3166084.

[86] S. Bruch, S. Gai, and A. Ingber, "An analysis of fusion functions for hybrid retrieval," *ACM Transactions on Information Systems*, vol. 42, no. 20, pp. 1–35, 2023, doi: 10.1145/3596512.

[87] Elastic, "Improving information retrieval in the Elastic Stack: Hybrid retrieval." 2024. Available: https://www.elastic.co/search-labs/blog/improving-information-retrieval-elastic-stack-hybrid

[88] H.-L. Hsu and J. Tzeng, "DAT: Dynamic alpha tuning for hybrid retrieval in retrieval-augmented generation," 2025, Available: https://arxiv.org/abs/2503.23013

[89] T.-Y. Liu, "Learning to rank for information retrieval," *Foundations and Trends in Information Retrieval*, vol. 3, no. 3, pp. 225–331, 2009, doi: 10.1561/1500000016.

[90] C. J. C. Burges, "From RankNet to LambdaRank to LambdaMART: An overview," Microsoft Research, MSR-TR-2010-82, 2010.