Evaluating Domain-Specific Topic Reduction for Sparse Vector Document Retrieval

Carson Irons

Advisor: Professor Boris Hanin

Submitted in partial fulfillment

of the requirements for the degree of

Bachelor of Science in Engineering

Department of Operations Research and Financial Engineering

Princeton University

May 2025

I hereby declare that I am the sole author of this thesis.

I authorize Princeton University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

_____

Carson Irons

I further authorize Princeton University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

Carson Irons

Abstract

This thesis investigates the limitations of current document retrieval systems and introduces an alternative architecture leveraging topic-level sparse indexing of contextual embeddings. This theoretical retrieval system seeks to achieve high computational efficiency through low latency and indexing overhead, while also achieving high semantic understanding and respecting local meaning and document cohesion. Additionally, the system supports scalable and context-aware document matching without reliance on user interaction data

In pursuit of these objectives, the system makes 2 key assumptions on the structure and content of documents within a chosen application domain. The first assumption is that documents can be broken into self-contained semantic components, the second assumes an ability to represent the application domain's distinct meanings as a finite, discrete set of topics.

At a high level, the proposed system aims to represent a document as a bag of topics, then apply sparse vector ranking algorithms at retrieval time. Topics are inferred by clustering the contextualized embeddings of semantic components within a learned embedding space.

The contributions of this thesis involve a review of existing retrieval methods, an outline of the proposed system's intuition and architecture, and an explorative implementation against a strategically chosen application domain. The thesis finds that standard embedding models (SBERT in this case) are insufficient for identifying application specific topics. Future work will focus on fine-tuning embedding models to better capture domain-specific semantics and fully evaluate the potential of this topic-based retrieval framework.

The thesis also provides the necessary tooling, for extension and modification of the retrieval pipeline. Namely, it supports the training and querying of the proposed retrieval system, while accepting custom implementations at each step.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

In the field of document ranking and retrieval, more commonly known as document searching, the central challenge is to balance computational efficiency with retrieval effectiveness.

Computational efficiency, refers to how fast a query can be performed. If a query can be performed quicker on average, your search system will have lower latency, which means the time between making the query and receiving the result will be lower, leading to a better user experience. Imagine, for example, searching on Google—if queries started taking even 1–2 seconds on average (seemingly still fast), most users would immediately notice. In fact, a 2009 paper by Jake Brutlag at Google found that a 100 ms to 400 ms increase in search latency would decrease the number of searches per day by 0.2–0.6% per user per day [3]. This, of course, is just one example of document retrieval. Most applications, especially those operating on internal documents (i.e., company documents, government records, etc.), have much higher tolerance for search latency. Regardless, in most cases, retrieval speed is critical for an effective system.

Computational efficiency also refers to the amount of memory necessary to index the documents in your retrieval system. Indexing is the process of strategically storing and structuring document metadata to enable faster retrieval. A common example of indexing is found in SQL. SQL, short for Structured Query Language, was developed by IBM in the 1970s[4], and allows for people to retrieve documents (rows of tabular data in this case) using a declarative syntax (declarative meaning you say what you want rather than how to get it)[17]. What made SQL great, in addition to its support for maintaining relationships between types of data, was its strong support for indexing. For example, if you anticipate a frequent querying of users based on the value of the `first_name` column, you could simply apply an index:

```
CREATE INDEX some_index_name ON users (first_name);[1]
```

Once applied, you can find documents based on this column with the following query:

```
SELECT * FROM users WHERE first_name = 'John';[2]
```

While you could technically make such a query without the use of the index in expression (1), the index allows the query in expression (2) to run in O(logN) or O(1) time complexity (depending on implementation rather than O(N). [17]

Big O notation follows the form $\mathcal{O}(\cdot)$ and specifies the order of growth with respect to the number of documents (N) in either runtime (how long an instruction takes to execute) or memory (how much memory is needed to store the data in a way that allows you to execute the instruction).[25]

Thus, in our previous example, to retrieve all documents that have `first_name = 'John'` (as seen in Expression (2)), a runtime of $\mathcal{O}(N)$ implies a need to check every document before finding the target.

For example, if you have 10 documents, it will take roughly $10 \cdot k$ milliseconds to execute for some $k > 0$, if you had 64,000,000 documents, it would take roughly $64{,}000{,}000 \cdot k$ milliseconds for the same k.

After applying the index in Expression (2) however, that same query (assuming base-2 logarithm) would take roughly $\log_2(10) \cdot c \approx 3.32 \cdot c$ milliseconds for some $c > 0$, and $\log_2(64{,}000{,}000) \cdot c \approx 25.93 \cdot c$ milliseconds for 64,000,000 documents.

In cases where we expect to query for documents based on `first_name`, this index will greatly decrease our query latency. However, as great as indexes are for speeding up retrieval speed, they also introduce their own overheads in terms of both write latency and memory.[17]

In order to maintain the index in Expression (1), you will need to set aside an extra chunk of memory for each inserted document. That is, you must maintain $\mathcal{O}(N)$ additional memory.[17] While slightly inconvenient, the amount of memory for each document is often much smaller than the memory required to store the document itself. For Expression (1), you must only store the value of the `first_name` column, not the entire document[17]. Thus, the memory for such an index is often a very reasonable tradeoff for the retrieval speed benefits it provides.

The second cost of maintaining the index in Expression (1) is that it slows down insertion speed. For each new document, you must not only add it to the database, but also restructure the index to reflect the new `first_name` being added[17].

This insertion might go from $\mathcal{O}(1)$ (append-based insertion) to $\mathcal{O}(\log N)$ when inserting into a tree-based structure (e.g., Red-Black Tree or B-Tree for ordering-based and primary-column-based indexes, respectively), or from $\mathcal{O}(1)$ to $\mathcal{O}(1)$ with a larger constant factor when inserting into a hashmap (for exact-match indexes).

The cost of this operation varies depending on the index type, but it should always be considered when designing or optimizing a search system with indexing.

As mentioned above, document retrieval systems must balance both computa-

tional efficiency and effectiveness. The former typically involves the use of (often creative) indexing strategies, to balance speed of reads and writes, and memory usage. The tradeoffs made here are largely determined by application specific use cases. For typical document retrieval applications, the priority is often placed on read (retrieval) speed with less focus on writes (which can be handled in the background) and memory (which is rarely a concern unless indexes must be maintained on RAM). While computational efficiency is a well defined objective, effectiveness is less clearly defined, and often more difficult to get right.

The effectiveness of a document retrieval system at its most basic level, is how relevant the documents it retrieves are to the specified query. This problem is not a concern in declarative query languages such as SQL, since the expected documents are well-defined, and any correct search system should return this set of documents every time. What happens, however, if your query language is less precise?

Retrieval effectiveness comes into the picture when a system must support queries in Natural Language. This problem is a major segment of the field of Natural Language Processing (NLP), and is what is actually referred to by the problem of document ranking/retrieval. In this context, the set of ideal documents is not closed-form, and thus a system must make educated guesses based on the presence of keywords in, and the semantic meaning of, the query.

There are several paradigms taken by modern natural language retrieval systems (from now on I will refer to these simply as retrieval systems). Each optimizes for certain key objectives. Sparse vector retrieval sacrifices semantic understanding for fast, but rigid, keyword based matching. Dense vector retrieval maintains solid query efficiency, and captures semantic meaning, but must sacrifice either local understanding (distinct ideas of the document) or document cohesion (how parts relate to the whole). BERT based approaches are the most effective, maintaining local and global meaning, as well as semantic understanding, but struggle greatly in computational

efficiency (and cannot scale to use in most applications). ColBERT, achieves comparable performance to BERT through pruning, but risks under or over pruning due to non-uniform densities in embedding space (more on this in next section).

1.2 Research Question

In this paper, we will explore a new paradigm for document retrieval that aims to achieve high computational efficiency, while also maintaining local meaning, document cohesion, and semantic understanding, especially for complex multi-part queries, and especially in specific application domains (more on this in later sections).

The primary goal will be to highlight the system's architecture and intuition, while also providing the tooling necessary to extend, improve, and run the system independently. In addition to this primary goal, we will perform a case study on this system, using a strategically chosen application domain, in an effort to walk through the process, evaluate assumptions, and examine queries in a real application.

Chapter 2

Literature Review

As hinted above there are several prevailing types of document retrieval, each with their strengths and weaknesses.

2.1 Sparse Vector Retrieval – BM25

One of the most simple and performant, yet surprisingly effective methods in document retrieval, is Sparse Vector Retrieval, often associated with the BM25 model[23] (more on this soon). Sparse vector retrieval refers to the process of representing documents and queries as sparse vectors (consisting of mostly zeros) that can be used as an analogue for similarity.

These sparse vectors act as an index, similar to the SQL example in expression (1). However, in addition to reducing latency, (by providing a compact and searchable analogue to the document itself), it also supports the use of natural language queries, making BM25, and other sparse vector methods valid Natural Language Document Retrieval Methods.

What does it mean to model a document as a sparse vector though? How are these vectors computed? How do they enable the use of natural language queries? To answer such questions, let's explore BoW (Bag of Words), its improvement TFIDF

(Term Frequency Inverse Document Frequency), then finally arrive at BM25 (Best Match 25) [23].

Modeling a document as a sparse vector involves choosing a set D of d characteristics that a document can possess, then constructing a d-dimensional vector $\mathbf{v}_i$ such that for each element v_{ij}, we have:

$$v_{ij} = \begin{cases} 0 & \text{if document } i \text{ is not characterized by } d_j \in D \\ > 0 & \text{if document } i \text{ is characterized by } d_j \end{cases}$$

with higher values of v_{ij} indicating a greater magnitude of characterization by characteristic d_j.

In practice, sparse vector retrieval methods typically choose D to be the set of all words that can appear in a document, which is referred to as the *dictionary*.

A simple Bag-of-Words (BoW) method might define $v_{ij} = \frac{\text{frequency of } d_j \text{ in document i}}{\text{\# of words in document i}}$, where d_j is word i in the dictionary.

To query documents in a BoW-indexed database, you would compute $\boldsymbol{v}_q$ from the query text. This allows you to rank document vectors $\boldsymbol{v}_i$ by how much keyword overlap exists between the query and document vectors.

In this simple form, the method retrieves documents in $\mathcal{O}(N)$ time complexity with an index storage complexity of $\mathcal{O}(ND)$.

The beauty of sparse vector representations however, is that they are "sparse". This sparsity allows for key optimizations to be made improving those metrics.

To speed up querying for large Ns, we can leverage inverse indexing to "prune" N to $n << N$. An inverse index works by using a map of D keys, one for each characteristic in a chosen characteristic set, and storing a list of all documents that are characterized by that key.[17] This means that in our bag of words example, rather than searching over every document in our database for the query "I like bananas", we can simply search over the documents contained under either the index "i", "like",

or "bananas". For large dictionary's this greatly decreases latency, as for most words (excluding common ones such as "the", "and", "I", etc.) a small portion of the N documents should contain any given word. One limitation which we will address later however, is that for lengthy documents and lengthy queries, the benefits of this optimization become less pronounced as documents fall into increasingly more buckets, and queries must make intersections across more buckets.

Similar to inverse indexing improving query latency, we can also decrease the storage overhead associated with our sparse vector representations using sparse vector compression algorithms. While the details of these methods aren't important, the general intuition is that in sparse vectors, there will be very long sequences of zeros (definition of sparsity) and can choose to compress these regions without information loss.

Consider, for example, a sparse vector that begins with a 1, followed by 200 zeros, then a 2, followed by another 400 zeros.

Storing this as a 602-dimensional vector,

$$\langle 1, 0, \ldots, 0, 2, 0, \ldots, 0 \rangle,$$

would require $602 \cdot c$ bytes, where c is the number of bytes required to store a single element ($c = 4$ for 32-bit or $c = 8$ for 64-bit values).

Instead, we could store the vector more efficiently using compression schemes such as:[26]

- **Run-length encoding:** $\langle (1, 1), (200, 0), (1, 2), (400, 0) \rangle$
- **Index-value pairs (coordinate list format):** $\langle (0, 1), (201, 2) \rangle$

These are both examples of *sparse vector compression*, which significantly reduce storage overhead, especially when working with large dictionaries and relatively small documents or queries.

While the optimizations help with computational efficiency, the bag-of-words model itself would struggle with effectiveness. The main shortcoming of this simple example is that the matching will be diluted by common stop words such as 'and', 'the' and 'or'. These will be prevalent throughout most documents and would lead to poor matching as a result.

A naive approach would be to simply remove these from our dictionary, but how do you choose which words to remove? Would you remove 'dog' or 'cat'? These words might appear more often than 'oscilliscope', for example, but they should not be completely omitted, as it would be tough to match a cat article to a cat query if you did. Additionally, if a query included both an uncommon word, such as 'oscilliscope', and a common word such as 'cat' we should ideally give more weight to documents that include 'oscilliscope' since it is the rarer word, and thus is likely to be very relevant to the query's intention.

To capture this idea of "rareness", we can use TFIDF (Term frequency Inverse Document Frequency) scoring. This model is similar to BoW, but punishes terms for being common across the corpus/database. In its simplest form, TFIDF is the product of TF(vj) and IDF(vj)[12]. Thus, each element in the document vector vi, would be

$$\text{TF-IDF}(v_j) = \text{TF}(v_j) \times \text{IDF}(v_j)[3]$$

Where,

$$\text{TF}(v_j, d_i) = \frac{\text{count of } v_j \text{ in } d_i}{\text{total number of words in } d_i}[4]$$

And,

$$\text{IDF}(v_j) = \log\left(\frac{N}{1+n_j}\right)[5]$$

where:

- N is the total number of documents in the corpus,
- n_j is the number of documents in which the term v_j appears at least once.

Expression 4 is the formalization of a BoW scoring, Expression 5 captures the penalty term, which will boost the influence of words like "oscilliscope" and reduce that of words such as "cat".

While an improvement over Bag-of-Words, there is still a critical limitation. What happens if one document contains 100 words, and another contains 100,000?

A larger document will naturally have more words and more occurrences of those words, leading to a higher term frequency (TF) score without a corresponding increase in the inverse document frequency (IDF) penalty. This imbalance greatly hinders the effectiveness of document retrieval across documents of varying lengths.

To address this, Stephen E. Robertson and Karen Spärck Jones introduced BM25[23], one of modern document retrieval's most well-researched and widely adopted sparse vector retrieval models. It sees extensive use, even 30 years after its conception.

BM25 builds upon the TF and IDF metrics outlined above, but enhances the scoring process by also accounting for document length. Each document is represented as a term frequency (TF) vector, similar to the Bag-of-Words (BoW) model. In addition, BM25 maintains a global inverse document frequency (IDF) vector—much like TF-IDF—as well as several corpus-wide statistics:

- N: the total number of documents in the database,
- $\mathbf{n}$: a vector where element n_j is the number of documents in which characteristic v_j occurs at least once,
- L: the average length of documents in the database,

- **l**: a vector where element l_i is the length of document i.

BM25 uses similar sparse vector optimizations to BoW (such as inverted indexing and compression), and applies the following scoring to a query-document pair [23]:

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

The IDF term captures the commonness penalty described in TFIDF, while also accounting for relative document length in addition to term frequency for a given document. This method works exceptionally well in practice, and is the default scoring model used in Lucene, an open source search engine software library that is extensively used at scale today, and is the engine behind the popular full-text search engine elastic search.

(1) Even though BM25 does a great job at capturing characteristic overlap between queries and documents, it is still limited by its choice of dictionary. While word based dictionaries are the most intuitive and effective dictionary for natural language queries, the resulting bag of words representation of documents prevents any attempts at semantic understanding.

2.2 BERT – Bidirectional Encoder Representations from Transformers

In 2018, Ashish Vaswami et. al. released modern AI's most pivotal paper: "Attention is all you need" [27]. In it, they proposed the transformer model which massively improved the scalability and context window of neural networks, especially in natural language processing. The transformer model quickly replaced Recurrence Neural Nets (RNNs) as the prevailing architecture for NLP. Traditional RNNs suffered from

several issues. They relied on sequential processing of tokens, where each token would gradually alter the state, this lead to scalability bottlenecks, as computations could not run in parallel. The effectiveness of relationship modeling was also lack-luster, as tokens would only know about preceding tokens, and long distance interactions were often diluted in favor of interactions with immediate neighbors. The Transformer erased all of these shortcomings at once. Its multi-headed self attention mechanisms allowed for parallelization, bidirectional encoding of relationships (tokens were influenced by tokens before and after it), and capturing of long distance interactions. Big words aside, this architecture laid the groundwork for rapid advancements in the AI space, the most influential being BERT.

BERT stands for Bidirectional Encoding Representations from Transformers, and was proposed by Google in November of 2018[6]. The aim of the model was to leverage the Transfomer model to generate highly context aware representations of documents, which could be used in downstream applications (such as Generative Pre-trained Transformers GPTs).

To accomplish its goal, BERT used the encoding architecture from the Transformer model. This encoding architecture goes something like this[27]:

1. Break query text into word/sub-word tokens.
2. Map tokens into high dimensional embeddings such that tokens that often appear together in language are close to each other in this high dimensional space. Also append the token's position in this vector.
3. Apply K (number of heads in the layer) independent activation maps (called heads, similar to those in CNNs) that output a relevance score for each token in the query text. For a given head, each query token will generate a relevance score for each other token including itself.
4. Feed the generated activation matrices through a neural network which generates a shifted version of the initial embeddings such that

their position characterizes their meanings with respect to the other tokens in the query text.

BERT takes this a step further, and builds a deep transformer network. Traditionally, it stacks 12 transformer layers on top of each other in order to capture increasingly nuanced relationships, and allow for future fine tuning, similar to how deep CNNs capture increasingly nuanced patterns in images as more layers are applied.[6]

Additionally, it introduces a special CLS token. This token serves as an unbiased token prepended to the query text. It undergoes the same multi headed attention mechanisms as the other query tokens, and synthesizes the interactions across the 12 transformer layers, acting as a highly context aware representation of the query text.[6]

This CLS token is the output of the BERT model, and is used as the input for downstream tasks such as sentiment analysis, chat bots, and even document retrieval. How convenient!

2.3 Dense Vector Retrieval – Semantic Search

With the proliferation of the BERT model throughout the NLP space, several new search paradigms emerged. The first of these is dense vector retrieval, commonly referred to as semantic search.[14]

Dense vector retrieval approaches the document retrieval problem from a completely different perspective than its sparse predecessor. It has its own intuition, optimizations, use cases, and limitations. While sparse vector retrieval methods such as BM25 rely on the presence and frequency of an enumerable set of characteristics (i.e. words), dense vector retrieval looks for documents that have similar meaning to the query. The definition of meaning is subjective, but in the context of dense vector retrieval, it means how close a query's embedding is to a document's embedding. As

mentioned above, an embedding is a representation of a piece of text as a high dimensional vector, such that two text embeddings are closer if the two pieces of text have a higher chance of being used near each other in natural language. Thus, at a high level, dense vector retrieval searches for documents whose vectors, generated from the same embedding model as the query's vector, have the lowest cosine distance (dot product over magnitudes) from the query vector. In other words, it finds the most semantically similar documents, agnostic of a set of enumerable characteristics.[14]

As the name suggests, dense vector retrieval models represent text as dense vectors (embeddings). This prevents them from utilizing the sparse indexing and compression algorithms enjoyed by their sparse predecessors. To circumvent this, dense vector retrieval uses their own optimization techniques to increase scalability by leveraging spatial indexing.

Spatial indexing, commonly associated with KD trees, is the process of structuring data in a way that enables efficient spatial queries, such as K nearest neighbors, and enables semantic search at scale.

While KD trees are most commonly associated with spatial indexing, they struggle with high dimensional vectors due to the curse of dimensionality[1]. This strategy segments space into a series of hyper-rectangles by partitioning one dimension at a time, and prunes points within these hyper-rectangles (more points in a hyper-rectangle = more points pruned per operation). In low dimensions, far away points can only vary by a few dimensions, thus partitions have a higher chance to split them with a random axis partition. However, as points reach higher dimensions, the variation within a single dimension can be small, even if two points are far away. This means that pruning is less effective, and runtime approaches O(N) complexity, rather than the O(logN) complexity of its low dimensional counterparts.

To circumvent this curse of dimensionality, a series of modern indexing algorithms have emerged. Some, such as KMKNN (K-means K-Nearest-Neighbors) attempt

to preserve complete accuracy, leveraging the triangle inequality to prune centroids rather than hyper-rectangles[10]. Others relax this correctness constraint, and perform approximate KNN (i.e. Facebook's FAISS[11], Google's ScaNN[13]) which blend various partitioning techniques to give probabilistic guarantees on accuracy without the overhead of precise algorithms. Regardless, the important thing is that modern spatial indexing algorithms enable low latency queries, even in high dimensions, making dense vector retrieval highly performant in practice.

(2) One of the major challenges of dense vector retrieval however, is a tradeoff between preservation of local context and full-document cohesion.

The reason for this, is that when embedding a piece of text, the resulting representation is an aggregation of the text's distinct ideas, into a single n-dimensional CLS vector (same one from BERT). This means that as document size increases, the model must capture increasingly large context windows, with more diversity in meaning and interactions.

Consider for example, the concept of full bidirectional self attention. This concept, fundamental to the transformer architecture, and by extension the BERT model, involves (at an abstract level) linear combinations of each input embedding, and thus, the more tokens interacting across multiple layers, the more diluted the resulting CLS vector.

For this reason, specialized models learn to prioritize increasingly generalized patterns at the expense of local token interactions. While this phenomenon isn't as pronounced in models such as SBERT [21], which specialize in embedding sentence length blocks of text, models such as BigBird[29] and Longformer[2], that operate on documents such as paragraphs, pages, or even entire articles/books, must inherently make this tradeoff. Thus, as document length increases, embedding models choose to encode abstract concepts, while sacrificing local meaning and nuance.

2.4 BERT Based Retrieval – Cross Encoders for document ranking

Today's most effective document retrieval models are BERT based cross encoders (ie MonoBert[7], SentenceTransformers Cross Encoder[22]). These models leverage cross attention to capture nuanced interactions between documents and queries. At a high level, this process:

1. Concatenates query and document together into a single token string. Ie. [CLS]query[SP]document[SP].
2. Passes this token string through a fine-tuned BERT model which encodes information on match quality into the CLS token.
3. The CLS token is passed through a linear decoding layer, to generate a document-query relevance score.
4. Documents are ordered by these relevance scores.

This retrieval paradigm has proven to be the most effective, by far, by several accepted benchmarks[16].

The model however, despite its effectiveness, is very computationally expensive. This is because for each of the N documents in the database, it must apply several transformer layers, each with a runtime complexity of $\mathcal{O}((Q + D)^2)$, where Q is the length of the query and D is the length of the document[16].

As a result, Cross-Encoder-based retrieval models have limited usability in production settings, since their latency:

1. Grows linearly with the number of documents N (as they do not support sublinear-time retrieval optimizations).

2. Grows quadratically with document length D at runtime (unlike dense vector retrieval methods, where $D \gg Q$ and documents are not encoded at query time).

This makes Cross-Encoders poorly suited for scenarios involving either large document collections or long individual documents.

Despite a lack of scalability however, cross encoders are highly effective at re-ranking promising documents, retrieved by faster sparse or dense vector models[16]. This ensures that its scalability constraints are bounded, while still leveraging its high effectiveness.

2.5 ColBERT – Late Interaction is the Best Interaction

In an effort to bridge the gap between the speed of earlier methods and the effectiveness of shiny cross encoders, Khattab & Zaharia 2020, released ColBERT (Contextualized Late Interactions over BERT)[16]. This model brought to Cross Encoders, what Inverse Indexing and Spatial Indexing brought to BM25 and semantic search, pruning. ColBERT selects promising documents based on token-level semantic similarity before performing cross encoder based reranking. In practice, this improves retrieval speeds by over an order of magnitude, enabling a much wider range of applications.

While at the cutting edge of document retrieval, it is still almost an order of magnitude slower than BM25 and semantic search[16]. Additionally, since it only considers the K most relevant documents to each query token, it runs the risk of omitting relevant documents when N (number of documents) is large. This is because, if there are a lot of highly relevant documents to a given query token, only the top K will make the cut. By increasing K this concern can be mitigated, but this will start to cut into the performance benefits that make ColBERT great.

Chapter 3

Approach and Intuition

As highlighted above, document retrieval cares about two things: Effectiveness and Computational Efficiency. Methods such as BM25 and semantic search demonstrate high efficiency, but have tradeoffs in effectiveness highlighted in observation 1 and 2. BERT based Cross Encoders, while highly effective, struggle with computational efficiency in practice, and even their optimized successor ColBERT is an order of magnitude higher in latency than baselines (especially as query length grows).

How do we reconcile these two objectives to achieve low latency queries without sacrificing semantic understanding (as in BM25), and without sacrificing local understanding or document cohesion (as in dense vector retrieval)? Further, can we maintain these objectives while supporting long, multi-topic queries, and even document to document matching, without relying on slow cross-encoder based models?

To reconcile these competing demands–namely, balancing effectiveness and efficiency while accommodating long, nuanced queries—I propose a system that, under a constrained application domain, represents a document as a bag of domain-specific topics, chosen from a discrete topic space that is learned from a set of application specific documents. This discretization of domain specific semantic meanings (topics), if possible, will allow for the use of performant sparse vector retrieval algorithms,

without the rigidity of a word centric characteristic set.

Further, I propose that by collecting topics across the entire document, we can ensure document cohesion, and that by extracting topics via locally contextualized passage embeddings, we can similarly preserve local understanding.

This system makes two key assumptions, each of which constrain its effectiveness to specific application domains, where document meanings and structures follow somewhat predictable patterns. These assumptions are:

1. **Document segment-ability**: Assumes that within a chosen application domain, the majority of documents can be meaningfully broken into semantically homogenous (of a single meaning) and semantically coherent (possessing all context needed to convey a single meaning) chunks in a computationally feasible, and discoverable way. (In other words, we can perform effective semantic chunking).

2. **Topic Reducibility**: Assumes that within a chosen application domain, the set of meanings which can be conveyed by a document chunk can be mapped to a learnable, and finite set of topics. (We can define, learn, and classify chunks into a meaningful characteristic set).

Given these assumptions hold within a chosen application domain, I propose a document retrieval system consisting of 4 steps.

3.1 Document Segmentation

The first step of this system involves breaking a document into a set of continuous, semantically homogenous, and semantically coherent sub-sections—we will refer to these sub-sections as Semantic Components.

1. Continuity: Ensures local structure is preserved, and enables transformer based contextualized embedding models to accurately represent the component's intended meaning.

2. Semantic homogeneity: Ensures that a semantic component can map to a single topic in the discretized characteristic set.

3. Semantic Coherence: Ensures that a semantic component possesses the information needed for a contextualized embedding model to properly understand and represent its meaning.

The method with which to do this should depend on the document structure within a chosen application domain. The problem of document segmentation is very challenging in the general case, as naturally, documents will have non-uniform semantic densities.

Consider for example, a novel versus a resume. A novel might have semantically coherent and homogenous sub-sections that are paragraphs or pages in length. A resume however, should realistically convey its semantic components within a single sentence or bullet point. This non-uniformity of semantic density, or how much text should constitute a semantic component, is highly difficult to work with, and is one of the reasons generalized semantic chunking is such a difficult task. Thus, in a suitable application domain, non-uniformity of semantic density should be minimized.

Regardless of choice of application domain, it is necessary to implement logic, with which to extract semantic components from a piece of text, i.e. a chunking method. Various methods exist of differing degrees of complexity. Some involve fixed size chunking, aiming to break off chunks of a fixed character length. Others use the location of delimiters such as periods and newlines to break documents on a sentence or paragraph level. Some experimental methods even leverage machine learning to choose breaks based on changes in meaning, which while promising for our use case,

is still an open area of research [28] [30].

3.2 Semantic Component Representation

The next step is straightforward and well researched, and involves representing a semantic component to be machine interpretable, while preserving its meaning. As mentioned above, BERT based embedding models are well suited for this task.

The choice of model depends highly on the choice of chunking method during the document segmentation step. For example, if semantic components are roughly of sentence length, SBERT (Sentence-BERT)[21] would be an intuitive choice, if paragraph length or longer, models such as Longformer[2] or Big-Bird[29] (embedding models trained to represent long pieces of text) are better suited.

3.3 Topic Reducibility

Once we have represented our document as a set of semantic components, then transformed these components into context aware embeddings, we must define a set of topics to make up our characteristic set.

To do this, I propose a discretization of our embedding space into a set of K neighborhoods (topics), such that if an embedding i maps to a topic j, we can say that the semantic component, represented by embedding i, is of topic $t \in T$(discrete).

The embedding spaces generated from models such as SBERT have been shown to perform strongly in clustering tasks and have been shown to separate distinct ideas very strongly without labels[21], suggesting that clustering will be a promising discretization approach.

One difficulty however, is that strong clustering performance requires that embeddings are generated relative to the definition of "meaning" for a task. To achieve this, fine-tuning of the embedding model at this step will likely be necessary.

In our context, the need for fine tuning arises from the fact that the distribution of meanings (and thus of the embeddings in embedding space) will favor certain regions (or neighborhoods in embedding space), since it is presumed the distribution of meaning in an application domain will not mimic the distribution of meaning in general language. Additionally, each document in the application domain will share the common context of being part of the application domain. This will presumably lead to an embedding space characterized by semantic hotspots and semantic deserts, leading to more pronounced outliers within deserts, and tightly knit, but sufficiently distinct ideas within hotspots.

To illustrate this, consider the application domain of resumes. Here, it is unlikely that semantic components will map to regions of embedding space associated with dogs or cats (unless they are applying for a veterinary role). However, it is similarly very likely that regardless of industry, regions associated with characteristics and skills (ie leadership, communication, spreadsheets, etc.), will be highly utilized. These highly utilized regions in embedding space are the aforementioned semantic hotspots (regions in which the embeddings of semantic components commonly map to), and the areas which see infrequent use, semantic deserts.

3.4 Sparse Vector Retrieval (again)

Now that we have represented our documents as bags of topics from our discrete and finite topic set T, our retrieval problem reduces to sparse vector retrieval over a characteristic set T. We can apply sparse vector indexing and compression techniques to enhance performance, and leverage BM25 (or other sparse vector techniques) for document ranking.

Thus, assuming the assumptions of document segmentability and topic reducibility hold for our chosen application domain, and assuming that our embedding model has

been fine tuned to alleviate semantic hotspots and deserts, we have a retrieval system that is:

1. Highly Performant: Latency is equal to that of BM25 + Representation(Q) where Representation(Q) = chunk(Q) + embed(Q) + classify(Q).
2. Semantically Aware: Since matching topics imply semantic similarity.
3. Preserving of Local Meaning: Each topic represents a fully contextualized semantic topic.
4. Preserving of Document Cohesion: Topics are represented as an aggregation of their local meanings.

These are all great, but we should also note its predicted limitations:

1. Non-Generalizable: Relies heavily on application domain specific fine tuning and implementation choices.
2. Challenging Sub-Problems: Effectiveness is tied directly to the minimum of the effectivenesses at which we are able to solve the document segmentation and topic reduction subproblems.
3. Constrained Application Domains: Domains with high variances in semantic density are more difficult.

Chapter 4

Implementation

4.1 Choosing an Application Domain

For this paper, I will be working with an application domain spanning plain text person descriptions that might be found in social media bios or dating applications. These descriptions will be of roughly paragraph length.

The reason for choosing this domain is twofold. First, this is a highly motivated space for document ranking/retrieval, as the user matching problem is central to various social media and dating apps. While keyword-based search, and matching algorithms tied to user behavior loops, mostly satisfy the requirements of these applications, the ability to query for closest matching person descriptions using multi-part plain text queries would be an interesting use case. Additionally, description-to-description matching (since matching can be implemented by using a document as a query), if effective, would enable scalable, user feedback agnostic recommendations. As an added bonus, recommendations and search results would be highly transparent, as matching topic labels could be tied back to their semantic components and placed side by side.

Second, paragraph length person descriptions follow similar structure to our pre-

vious example of resumes. Namely, by aiming to convey a lot of non-causally related ideas in a short amount of space (ie independent characteristics, interests, values, etc), documents in this domain often consist of relatively independent, semantically self-contained sentences.

As a result, I assert that this space will allow for effective sentence level chunking, greatly simplifying the document segmentation subproblem, and allowing for an easy choice of SBERT [21] for representation. Additionally, things like interests, values, and personality traits (all commonly found in short-form person descriptions) are highly enumerable. This, I assert, should make this application domain highly suitable for satisfying the topic-reducibility assumption.

4.2 Dataset Generation

As might be expected, datasets consisting of plain text person descriptions are difficult to come by. However, a dating app called OkCupid published an anonymized dataset on Kaggle[19], consisting of 60,000 rows of csv formatted user profile data. For a given user, the dataset contains information on things such as their age, gender, occupation, and certain lifestyle habits, as well as up to 7 essay responses to unknown prompts. Given this extensive data and access to free-form essay responses, I opted to generate a dataset of synthetic user descriptions, tied to the information and writing style of each user.

To do this, for each person, I generated a rough blurb using their csv data. For data included in every row, (like whether they like dogs or drinking), I used probabilistic omission to ensure sufficient variation in included details (otherwise each person would mention each of these in their description). The person's age and career was always included, as I felt it was relevant to how they might describe themselves. Next, I appended the essays to the description to give insight into the person's writing style,

personality, and general 'vibe'. As an added bonus these essays often reveal relevant information about the person.

From here, I constructed and passed prompts to the GPT 4o-mini model by OpenAI to make descriptions realistic, and to spruce them up with reasonable guesses about other characteristics. The prompt generation logic can be found on Github (link included in appendix). Anchoring person description generation to real user data, ensures that the synthetic dataset is representative of an actual distribution of characteristics, and preserving natural variance between people.

4.3 Research Runner

For the purposes of extensibility and reproducibility, one of the major contributions in this paper is a utility called ResearchRunner, which can also be found on Github. This utility implements the retrieval system with end-to-end batching, persisting, interruptibility, and visibility to abstract system complexity. It also allows for quick iteration and experimentation using different combinations of implementations for document segmentation, representation, and topic classification.

To train and query against an implementation of this system, simply implement the `ChunkingAgent`, `EmbeddingAgent`, and `ClusteringAgent` interfaces (see below). Then, initialize a `ResearchRunner` instance by passing these agents into the constructor, and call `run_research()`.

The system will log progress throughout the training process. Once completed, you can call `query()` to retrieve the top results from the utility.

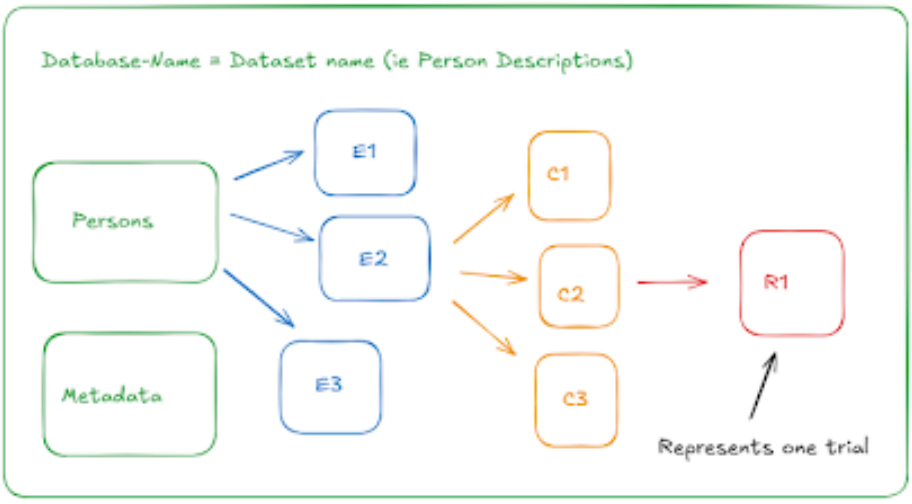
This utility is configured to pull from a persons table, but can be modified according to other application domains. To replicate trials on the person description application domain, you can download the full project zip, and use datasetGenerator to ingest the generated dataset into the pipeline.

Given the amount of iteration and trial and error, this utility was necessary to avoid re-coding the pipeline each time. Given the size of the data and duration of computations, interruptibility was necessary to ensure that pipeline state was never corrupted by crashes or interruptions. Logging was added as a quality of life feature, and a sanity check to gain visibility into the pipeline's progress during training.

The following sections break down the architecture of the workflow, along with the interfaces for each of the aforementioned agents.

4.4 Database Organization

To ensure that results and intermediate computations are saved and resistant to crashes, I leveraged SQLlite for data persistence. To ensure that intermediate results are easily retrievable and that results are tied to specific trials and implementation choices, it was important to enforce strict table naming conventions. These choices allowed for easy organization, and reuse of intermediate results. Especially for reuse of the embeddings tables which are computationally expensive to generate. A diagram for this can be found in Figure 4.1



Persons ~ Stores the text descriptions of M persons
Metadata ~ Stores metadata on the status of each trial. Used to resume training progress in the event of a crash

Embeddings ~ Stores the component vectors across all M persons
Table name: Embeddings_ChunkingMethod_EmbeddingModel)
Quantity: One for each chunking->embedding implementation

Centers ~ Stores the K cluster centers from topic identification
Table name: Centers_ChunkingMethod_EmbeddingModel_ClusteringMethod
Quantity: One for each trial

Results ~ Stores the M result vectors, one for each person
Table name: Results_ChunkingMethod_EmbeddingModel_ClusteringMethod
Quantity: One for each trial

Figure 4.1: Database Organization

4.5 Modular Agents

In addition to careful database organization, the use of modular agents further improves the speed and ease of iterations. The process uses 3 core agents:

Chunking Agent: Handles the chunking of a particular description into its semantic components.

```
## Chunking agents implement segmentation step
class ChunkingAgent(Protocol):
    name: str

    ## Extract semantic components from text
    def chunk(self, raw_text: str) -> list[str]:
        ...
```

Embedding Agent: Handles mapping a textual semantic component to N-dimensional vector space to attain a component vector.

```
## Embedding agents implement the representation step
class EmbeddingAgent(Protocol):
    name: str

    ## Generate a text embedding for a semantic component
    def embed(self, raw_text: list[str]) -> list[list[float]]:
        ...
```

Clustering Agent: Handles the segmentation of embedding space into a set of K topics as well as the generation of result vectors. (A result agent might be added if alternative modes of result vector generation are explored).

```
## Clustering agents implement the topic reduction step
class ClusteringAgent(Protocol):
    name: str

    ## Pass the embeddings object (used by pipeline to access
       embeddings table
    def pass_embeddings(self, embeddings: Embeddings):
        ...
    ## Learn topics, pass_embeddings must have been called
       previously
    def train(embeddings: list[list[float]]):
        ...
    ## Get sparse vector representation for component
       embeddings
    def generate_result(self, person_embeddings: list[list[
       float]]) -> list[int]:
        ...
    # Classify an embedding into a topic
    def topic_map(embedding: list[float]) -> int:
        ...
    ## True if train has been called previously
    def is_finished_training() -> bool:
        ...
```

4.6 Running a Trial

Once an implementation for each agent has been chosen, we can execute the entire process with a single function call. In the process all intermediate calculations can be retrieved using the table naming conventions specified above. Figure 2 shows the workflow for running a trial given a 'Persons' table, 'Metadata' table, and ResearchRunner (The class that implements the workflow) object.

Note that the centers table must be maintained by ClusteringAgent implementation.

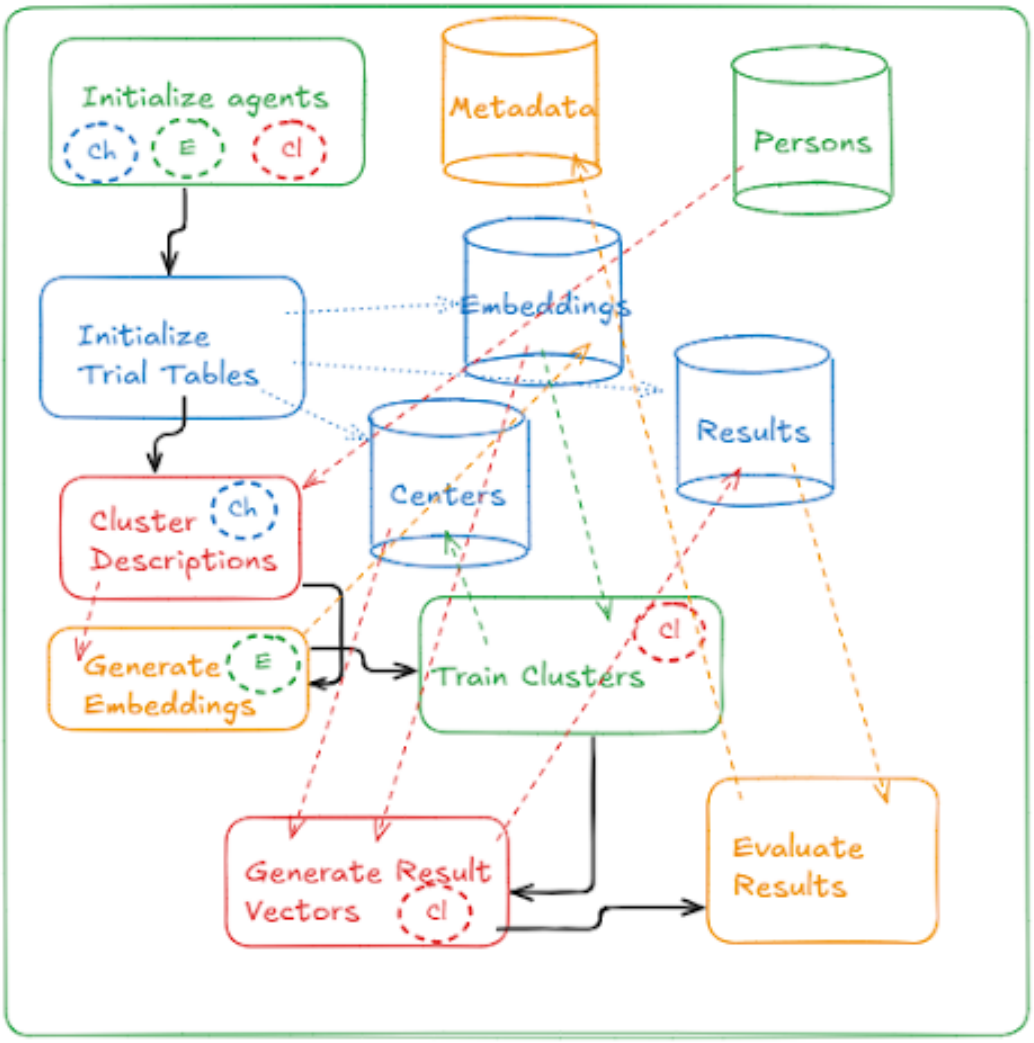


Figure 4.2: Architecture of Research Pipeline

Chapter 5

Literature Review II

For the following sections, we will be exploring the subproblem of topic reduction, involving a discretization of embedding space. As previously mentioned, this will involve a clustering of the realized embedding space informed by the distribution of semantic component representations in our dataset.

5.1 Clustering

The field of clustering involves itself with learning optimal criteria for defining regions in n-dimensional space in a way that places similar data in the same bucket. Similarity is a subjective term, but is, in most clustering related contexts, related to the distance (ie euclidean, cosine, etc.) between two points in space. Various methods have been developed to solve this problem, each with their own approaches to parameterization, and their own assumptions on the optimal shape of realized clusters. [9]

As noted above, modern clustering algorithms can be categorized by their approach to deciding the optimal number of clusters. The most standard and efficient way to do this is to let the user specify this via a hyperparameter. Methods that use this approach are referred to as parameteric [9]. Hierarchical clustering however, learns the optimal number of clusters during training, but often comes at the cost of

computational efficiency[9].

A popular example of parameteric clustering is K-Means. This algorithm initializes a random set of cluster centers within the data's n-dimensional space, then over a series of epochs (passes over the training data), runs the following steps: [8]

1. For each point, assign it to the closest center by euclidean distance.

2. For each cluster, assign its new center to be the average (center of mass) of the points in the cluster.

This process converges to a set of centroid-shaped clusters, which make a reasonable attempt to minimize the sum of within-cluster variances (inertia). This method is computationally efficient, running in $\mathcal{O}(ndk)$ time, where n is the number of data points, d is the number of dimensions, and k is the number of clusters. Classification at inference time operates in $\mathcal{O}(kd)$ time which is highly performant. Its parametric status comes from the fact that number of clusters remains fixed, based on the user-specified initialization.

Some clustering algorithms however, choose to leverage hierarchical clustering. [9]. This means that the algorithm learns the optimal choice of clusters as it goes, and is best used when you are unsure of an optimal choice for the number of clusters. One of the most popular hierarchical clustering algorithms is HDBSCAN[18], which also differs from K-Means in the shape of its clusters at convergence. That is, rather than centroid shaped clusters, it learns clusters by density rather than nearest center.

To accomplish the intersection of hierarchical and density based clustering, HDBSCAN employs a very complex algorithm, which goes something like this[9]:

1. For each training point x, compute its core distance core(x), such that core(x)=distance to kth neighbor

2. For each pair of points a and b, compute its mutual reachability distance

MRD(a,b) = max(core(a), core(b), dist(a,b)). This ensures that nearby points in sparse regions are still treated as far apart.

3. Create a minimum spanning tree (graph with N-1 edges such that each point is connected via some path that minimizes the sum of edge lengths) using MRD as the distance metric.

4. Iteratively remove the longest edges, increasing the number of clusters.

5. Choose the most stable set of clusters, such that a cluster is more stable if it was able to survive a larger range of edge removals without separating into two components (cluster that splits off with at least p points, for some p).

This process will converge not to a set of centroids, of fixed quantity, but rather a set of flexibly shaped regions of tightly packed points.

The purpose of the mutual reachability distance is to ensure these clusters are based on relative distances of surrounding points, rather than pure euclidean distance of those points.

The iterative removal of edges ensures that outliers (points not sufficiently close to enough neighbors) are not assigned a cluster label (different from KMeans approach of assigning every point to a cluster).

The choice of the most stable clusters ensures that clusters can be of non-uniform density. In other words, it finds the points that have low internal distances, relative to their external distances.

To illustrate this, imagine three regions of points: Region A with density d_a, Region B with density d_b, and Region C with density d_c, such that $d_a > d_b > d_c$.

Now suppose that the mutual reachability distance between Regions A and B (i.e., the density of the region connecting them) is much higher than the inner density of Region C (i.e., d_c). If we were to stop the clustering algorithm at a fixed point, such

as after a specific number of edge removals, we would be unable to recover all three clusters.

This occurs because the algorithm removes edges in order of increasing distance (i.e., decreasing density). Therefore, it would first disconnect Region C from the combined structure of Regions A and B. However, since Region C is sparser than the connecting region between A and B, the algorithm would continue to remove edges within C, eventually dissolving it entirely as a coherent cluster. Only after this would the connection between A and B be broken.

Consequently, the algorithm is forced to choose between two suboptimal clustering outcomes: either treating all three regions as one cluster ($A \cup B \cup C$), or dividing them into just two clusters–either ($A \cup B$) and C, or A and ($B \cup C$). The desired solution with three distinct clusters (A, B, and C) cannot be realized unless the full hierarchical structure is considered.

HDBSCAN's stability based election will actually keep C intact since assuming it existed for some wide range of splits elsewhere, it would be deemed stable enough to remain a cluster, even after later splits eventually break it up. Thus, the realized clusters will be those that persist for the largest range of densities, rather than those that manifested at some fixed density level.

To summarize, clustering methods can be parameterized or hierarchical. Parameterized means we choose the number of clusters, while hierarchical methods dynamically choose the optimal cluster count based on some heuristic (ie stability in HDBSCAN).

Clustering methods also define similarity differently. Centroid based clustering assumes clusters are distributed around a center of mass, while density based clustering cares only about the relative closeness of a series of points compared to neighbors.

For parameterized, centroid based clustering, we can use KMeans. For hierarchical density based clustering, we can use HDBSCAN. Some methods flex other combina-

tions, (such as DBSCAN, HDBSCAN's parameterized predecessor), but for now we will stick to the paradigms explained above.

Chapter 6

Results

6.1 Embedding Space Discretization

With our newfound understanding of clustering, we must decide which is most practical for our application domain. Without fine-tuning, I hypothesized that the distribution of our embedding space would be characterized by a series of semantic hotspots and deserts. The hotspots would consist of multiple, ideally separate clusters (i.e. playing sports and listening to music are similar in that they are interests, but should be different in our context) and deserts would contain many outliers (i.e. someone who really likes oscilloscopes). Thus, while KMeans is the most practical computationally, and has shown promising results within fine tuned settings [21], I believe that HDBSCAN will be optimal for discretizing a non-fine tuned embedding space.

6.1.1 KMEANS

To start, I tested the pipeline with KMEANS clustering as the discretization approach. Given that fine-tuning is out of the scope of this paper, I was not expecting strong results due to the semantic hotspot/desert hypothesis. Regardless, I decided to learn an optimal cluster (topic) count for the KMEANS model. To do this, I trained

a series of KMeans models of different cluster counts, then graphed them against several common evaluation metrics.

Inertia

The most common approach to hyperparameter tuning in KMEANS is the elbow method, which involves plotting model inertia[15] against cluster (topic) count and looking for an elbow (step decrease in rate of inertia decrease). Inertia captures the sum of inner-cluster variance of a clustering. Thus, a higher inertia suggests that clusters are more sparse, while a lower inertia suggests denser clusters. The intuition behind the elbow method is that it is the place where additional increase in cluster counts yields less marginal decrease in inertia, so by stopping here, we often avoid over clustering.

As seen in Figure 6.1, our embedding space yields a strong elbow at around 150-200 clusters. This implies that in the optimistic case, there are 150-200 distinct meanings that can occur in our documents, or in the pessimistic case, 150-200 hotspots that each contain several related topics. Regardless, the presence of an elbow is a good signal of clusterability.

Silhouette Score

The next, arguably most important approach, is evaluation of the Silhouette Score[24]. This metric measures how separated a set of clusters are and is computed via the following formula:

$$\sum_i s(i) = \frac{\max(a(i), b(i))}{b(i) - a(i)}$$

A higher value (closer to 1) suggests that points are much closer to their own cluster than to points in other clusters. Similarly, a lower score suggests that points are closer to points in other clusters than those in their own cluster. A value of zero

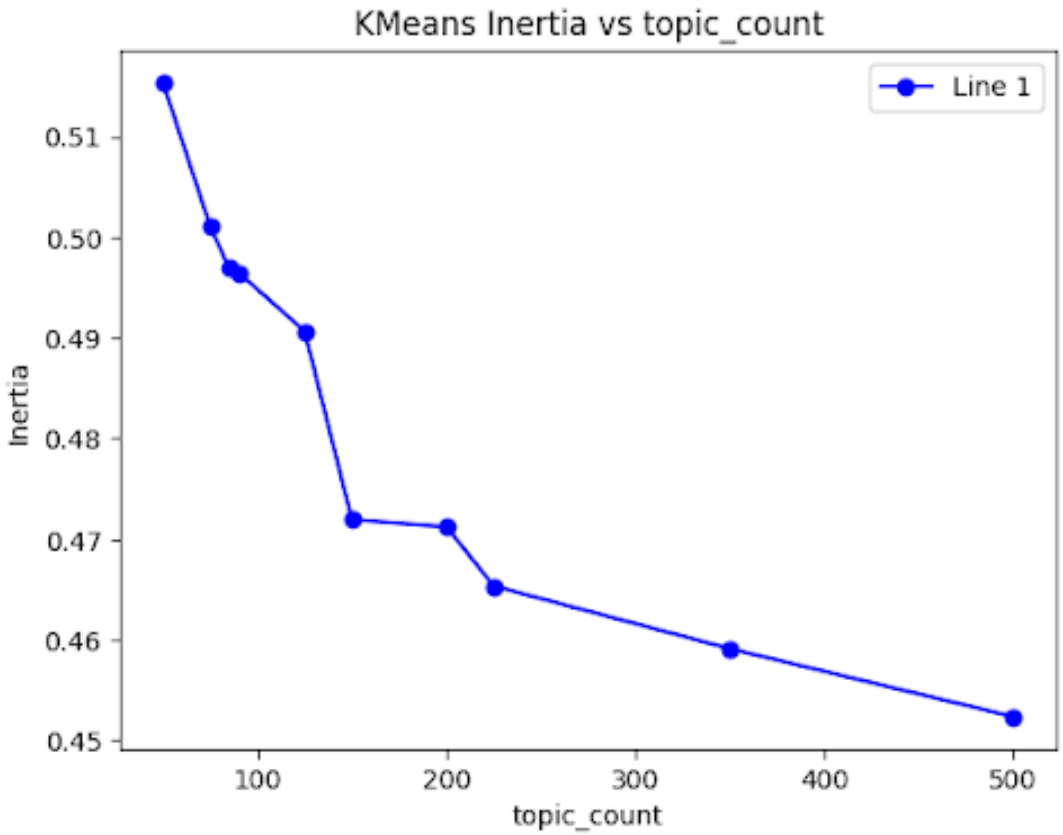


Figure 6.1: KMeans Inertia vs Topic Count

suggests that these distances are the same on average and typically means a point is on the border of a cluster.

As seen in Figure 6.2,The Silhouette score never exceeds 0.04 for $k > 75$. This suggests that the optimal number of clusters is significantly lower.

Under the assumption that our application domain should contain at least 75 distinct topics (an assertion I stand by), this observation indicates that our embedding space most likely consists of a series of semantic hotspots. Increasing k beyond 75 appears to arbitrarily fragment these regions rather than reveal additional meaningful structure.

This metric alone, I argue, disqualifies the efficacy of KMeans in a non-fine-tuned embedding space. However, for completeness, the Davies–Bouldin[5] and Calinski–Harabasz[**?**] metrics will be included in the appendix.

Figure 6.2: KMeans Silhouette Score vs Topic Count

This is further supported by the Figure 6.3 which shows a 3d UMAP reduction[**?**] of the embedding space under a KMeans (k=200) clustering.

6.1.2 HDBSCAN

While a poor Silhouette score suggests that non-fine-tuned embedding space does not properly cluster into exhaustive (including all points) centroid based clusters, there is a chance that strong clusters exist in non-centroid based shapes. To explore this, I attempted to run an HDBSCAN clustering on the data. Unfortunately however, given the non-batchability and memory intensive nature of the algorithm, I was unable to include more than 30,000 of the 400,000+ embeddings.

When using this subset, the clustering performance was not great. For multiple attempts, several clusters would converge to containing 10 or less points, even when

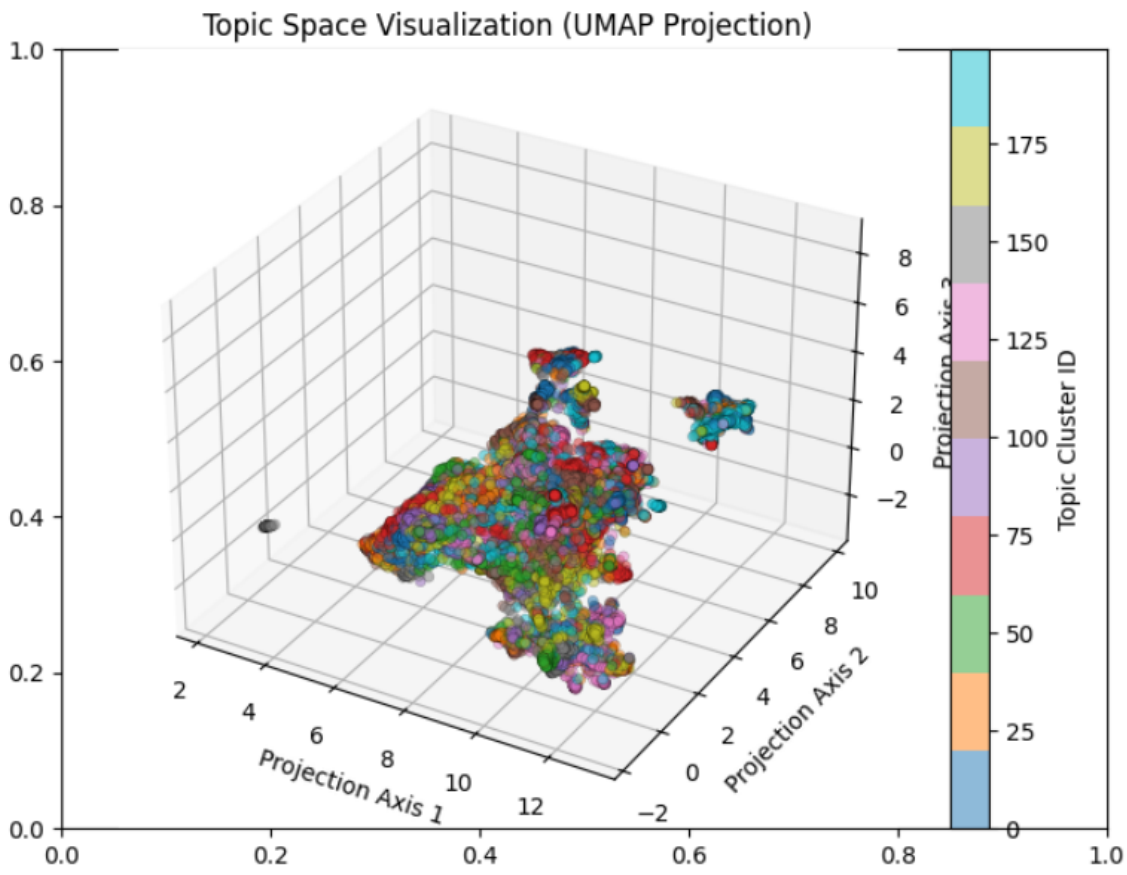


Figure 6.3: 3d UMAP of 768 dimensional Embedding Space with Realized Cluster Assignments

minimum cluster size (determines how many points must separate from a given edge removal to signal a new cluster) was set to 40 or higher. This suggests a high level of instability since larger clusters scarcely lasted long enough to be included in the final result.

Despite unpromising results, its possible that the undesirable clustering performance emerged from data omission caused by computational bottlenecks. Perhaps with more RAM, better HDBSCAN based clusterings could be realized, but this is purely speculative.

6.1.3 Takeaways

The above experiments highlight the necessity of a strong fine tuning of embedding models before approaching embedding space discretization. While this was expected to be the case from the start, the extent of fine tuning's effect was more drastic than anticipated. Despite this inherent structural limitation on topic classification performance, lets take a look at how it works in practice for completeness.

6.2 Single Topic Query Results

Query: "I like singing and dancing. Also I really want to do engineering"

Result: "I'm a 24-year-old female who values meaningful connections and enjoys meeting new people. Dancing is my passion and a vital part of my life, which I also share by teaching Polynesian dance to kids in the evenings. My dogs are my constant companions, and I love spending time outdoors, especially at the beach or park. Although I enjoy social drinking, I prioritize a drug-free lifestyle. Sports, particularly football, are a big part of my weekends. I've traveled to several countries, including Italy and Tahiti, and dream of exploring more of the world, with Samoa next on my list. I currently work as a receptionist but aspire to further my education with a master's in criminal justice or forensic psychology, aiming to counsel juveniles in the future. Music fuels my spirit, especially RB and island jams, and I cherish moments with family and friends. Most people compliment my captivating eyes."

→ Weird, matches music but doesn't find engineering...

Query: "Also I really want to do engineering"

Result: "I'm a 26-year-old woman working in the fast-paced realms of banking, finance, and real estate. With a passion for adventure, I relish exploring new restaurants, cultures, and music, often with a playful, slightly vulgar sense of humor. Volunteering in my community is vital to me, and I'm always eager to listen to the stories of others, especially those who've lived unconventional lives.

I'm a dedicated nerd at heart, often indulging in military history and strategy shows, while embracing my tomboy side through shooting, gaming, and regular workouts. Currently pursuing a degree in criminal justice, I'm excited about my upcoming graduation and the opportunities that lie ahead.

My interests extend to cinema, with a particular adoration for Edward Norton's films and classic rock, especially The Scorpions, who always brighten my day. Dancing is my ultimate stress relief, and Mexican cuisine holds a special place in my heart, reminding me of home.

Life is a journey for me where I frequently find joy in simple pleasures—like beach outings at odd hours, early morning fog views from the Golden Gate, and playful challenges in poker. I carry my phone, eyeliner, and a good excuse wherever I go, embodying my eclectic and whimsical nature. If you're up for adventure, there's always room for one more on my next big vacation."

→ It seems like engineering is being mapped with criminal justice for some reason...

Query: "I am an engineer"

Result: "Iḿ a 31-year-old male working in the science and tech field, with a strong passion for exploring free culture. I have a love for both dogs and

cats, viewing them as companions that enrich life. Rock climbing is my recent obsession; I seek out climbing spots in every city I visit while constantly on the move. I enjoy socializing over ping pong, mixing cocktails, and cooking up delicious meals. My tastes run towards books by David Sedaris, Christopher Moore, and classics like "Dune" and "Enderś Game," while I appreciate films like "Being John Malkovich" and "The Big Lebowski." Food is my greatest joy during travel; I find delight in trying local cuisines and experiencing new dining adventures. Iḿ driven by a thirst for knowledge and love engaging conversations with good people. I also relish the unpredictability of life, like the time I twisted my ankle while DJing. Overall, I'm always dreaming of my next travel destination and the amazing connections waiting to be made in those places."

→ Hmm, maybe it was matching based on a desire to pursue something, where pursuing engineering and criminal happened to make it in the same cluster... At the very least, the results were the same regardless of whether the query was "Also I want to be an engineer", "I want to be an engineer" "One day ill become an engineer" or "Engineering is my dream", but does change when you use "I am currently an engineer". This suggest strong semantic understanding and consistency, despite a non-ideal differentiation between wanting to become something versus being something, and a lack of differentiation in certain careers.

The next 4 results for "I am an engineer" included:

1. "I enjoy keeping up with technology" and "immersed in the science and tech industry, where I thrive on innovation"
2. "science and tech industry"
3. "world of science, tech, and engineering as a developer and business owner"

4. “innovation and problem-solving”

This suggests a decent clustering. Though it is not ideal that the choice of phrasing/intention makes the difference it does. It is also weird that pursuing criminal justice is so highly coupled with wanting to be an engineer.

6.3 Multi-Topic Query Results

Lets check out the real interesting question now... Multi topic queries => Matching. What if we use a document as a query??

> **Query:** “I’m a 27-year-old craftsman who finds joy in the details of construction and the beauty of the built environment. I see myself as a gentleman and a scholar, valuing kindness and intelligence. I’m currently learning TIG welding and appreciate the creativity it brings, even if it means sharing my workspace with an awful lot of donuts. When I’m outdoors hiking or camping, I can’t resist making quirky bird and alien sounds. My home is filled with succulents and prairie grasses, a testament to my passion for nature.
>
> I thrive on making people feel better and strive to be a good man in everything I do. My days often start with kale and coffee, with support from my grandmother fueling my pursuits. I cherish classic activities like playing board games, attending improv shows, and enjoying dinner with friends, often imagining how buildings can foster connections among people. I may be sweet and a bit odd, but I genuinely enjoy meeting new faces—let’s connect over a drink or meal and see where the conversation takes us.”

> **Result:** “I’m a 28-year-old tech enthusiast working in the computer hardware and software field. I’m a clean-living person who doesn’t engage in drug use

and values genuine human connections. My weekends are spent exploring every corner of San Francisco, indulging in new cafes and off-the-beaten-path restaurants. Climbing is a passion of mine; I enjoy it a couple of times weekly and love belaying friends. Apart from being skilled at fixing things, I have a knack for coding and baking.

I've honed the ability to fall asleep at will, a skill I'm quite proud of. My interests in literature lean toward Latin American fiction, and I aim to read one book from each Nobel laureate. Music is a vital part of my life: I always enjoy the classics like The Beatles and Pink Floyd but am also discovering new sounds. French cuisine tops my list, with a particular love for decadent dishes, and I'm not afraid to admit my soft spot for foie gras.

Good coffee, sunny weather, and great audio systems are essentials to my day. I often reflect on how language shapes my thoughts, oscillating between Spanish and English. My ideal Friday night involves good food, drinks with friends, and winding down with some quality music at home. I'm on a journey to find love, but my calculations so far suggest it's a challenging quest. I believe in the power of insight and kindness and always seek connections with like-minded individuals."

→ In this situation, we actually have a series of plausible matches. While I think we can definitely agree the clustering logic isn't perfect for this use case (to be expected without fine tuning), I think it shows promise in its ability to match a query and document based on the presence of multiple distinct overlapping characteristics.

Chapter 7

Future Work

7.1 Fine tuning

While this, somewhat naive implementation failed to achieve the desired retrieval performance within this chosen application domain, the potential of this system remains to be validated. It is very plausible that discretization of embedding space will be highly effective under a fine-tuned model

For the person description application domain, fine tuning would involve separating unique interests and characteristics from their generalized embedding spaces. This would alleviate counter-intuitive intersection such as the engineering and criminal justice example above. It would also involve a reinforcement of similar ideas to prevent loss of generality and semantic nuance in the matches. This step would depend on subjective definitions of similarness as whether singing and dancing should be related will be an impactful, but opinionated decision.

7.2 Enhanced Document Segmentation

For the scope of this paper, the challenge of document segmentation was skimmed over in order to focus on the Topic Classification (embedding space discretization) subprob-

lem. Our chunking strategy involved a simple sentence chunking based approach, but should hopefully be replaced by more sophisticated, context aware, semantic chunking methods. This will help in situations where an optimal semantic chunk can be smaller than a sentence, or span multiple sentences. Of course, this increases the complexity, but it will likely greatly expand the effectiveness and generalizability of the system as a whole.

7.3 Alternative sparse retrieval methods

The final, and arguably easiest improvement on the system realized above, is a choice in alternative sparse retrieval methods. I chose to use a simple Bag of Words (or Bag of Topics in this case) implementation. This was chosen because of the unpolished nature of this experimental implementation. I could used more advanced methods such as BM25 inorder to weight less frequent topics higher and longer documents lower, but as shown in above sections, the core limitation was in poor relationship modeling within the standard SBERT model. I reasoned that without a proper mapping of semantic components to reasonable topics, improvements at the final stage would not actually fix anything.

Chapter 8

Conclusion

In this work, we set out to explore a new paradigm for document retrieval that balances computational efficiency and retrieval effectiveness, particularly in domains where documents can be broken into semantically coherent chunks and mapped onto a finite, discrete set of topics. Unlike generalized approaches to document retrieval that rely on keyword-based matching or purely dense vector embeddings, our system aims to preserve local meaning and document cohesion by breaking text into semantic components, then clustering these components in the embedding space. This process effectively transforms each document into a bag of semantically encoded topics, that can be retrieved using computationally efficient sparse vector retrieval algorithms.

The experimental implementation, tested on a synthetic dataset of short, multi-topic person descriptions, demonstrates both the promise and limitations of this approach. On the one hand, our pipeline successfully retrieves documents with multi-faceted overlaps in characteristics and interests, underlining its potential for nuanced applications such as user matching or context-rich querying. On the other hand, the clustering results reveal that typical off-the-shelf embedding models (for example, SBERT without domain-specific fine tuning) may not segment meaning in ways that align with application specific search needs. In practice, it was determined

that functionally unrelated concepts (like 'engineering' and 'criminal justice') ended up clustered, further evincing the need for application specific fine-tuning. On the whole, this method, while conceptually appealing, is highly dependent on the effectiveness of semantic chunking and robust and predictable topic identification, which are difficult to get right in practice.

Nevertheless, the overall framework opens the door for future research and real-world deployment in settings where documents are structurally consistent and meaning can be discretized in a predictable way. Potential improvements include fine-tuning embedding models for the specific distribution of semantic concepts in a chosen domain, exploring more adaptive or machine-learning-based chunking strategies, and replacing simple "bag of topics" retrieval with more advanced weighting methods (such as BM25 or other scoring functions) to capture rarity or specificity of topics. Additionally, incorporating deeper dimensionality reduction methods (especially powerful for constrained application domains), advanced clustering heuristics (depending on K-Means performance in fine-tuned embedding spaces), or hybrid solutions (e.g., partial cross-encoder re-ranking on top of topic-based filtering) can further improve both speed and precision.

Taken as a whole, this thesis contributes an architectural blueprint and an extensible software pipeline that researchers and practitioners can adapt to a broad range of domain-specific retrieval tasks. While the high-level vision of scalable, semantically aware retrieval remains an aspirational goal, the iterative experiments, tooling, and discussion presented here, shed light on both the practicality and the inherent challenges in building a system that preserves local context, document cohesion, and semantic nuance, without sacrificing the real-world performance requirements essential for most retrieval applications.

Chapter 9

Appendix

A Unused Graphs

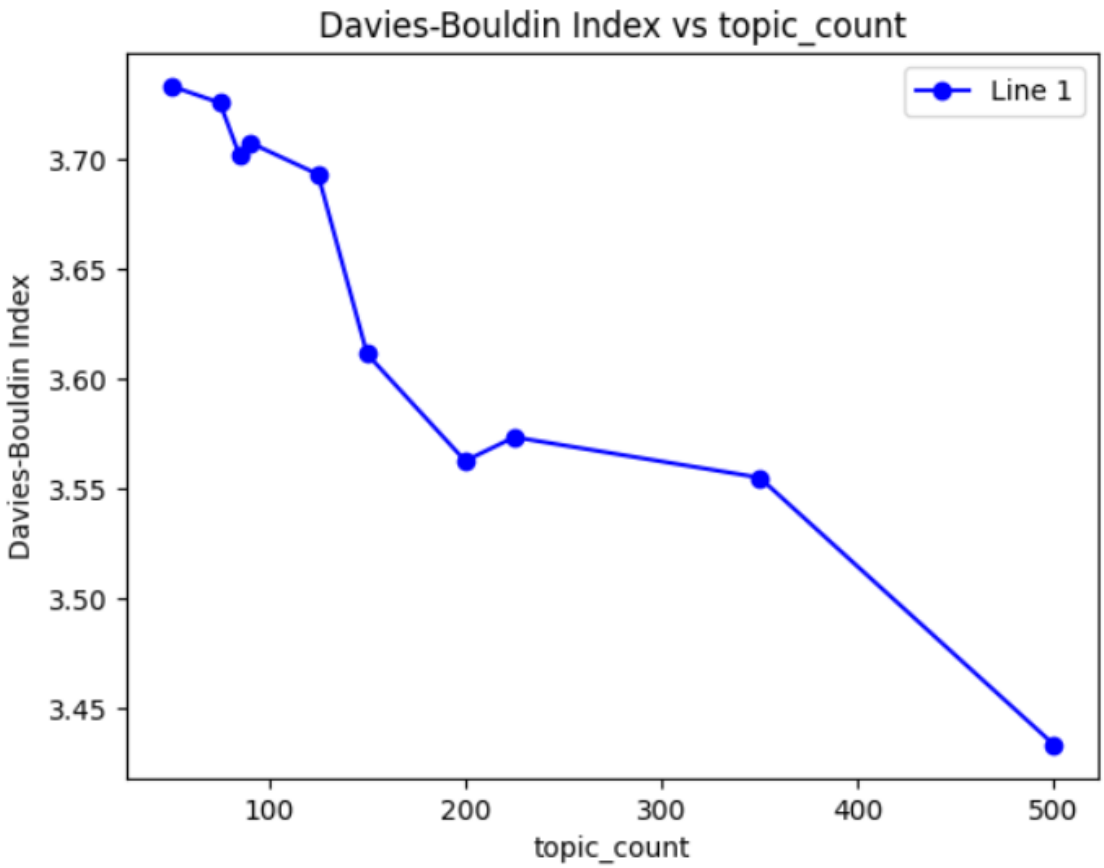


Figure 1: Davies-Bouldin Index versus Topic Count

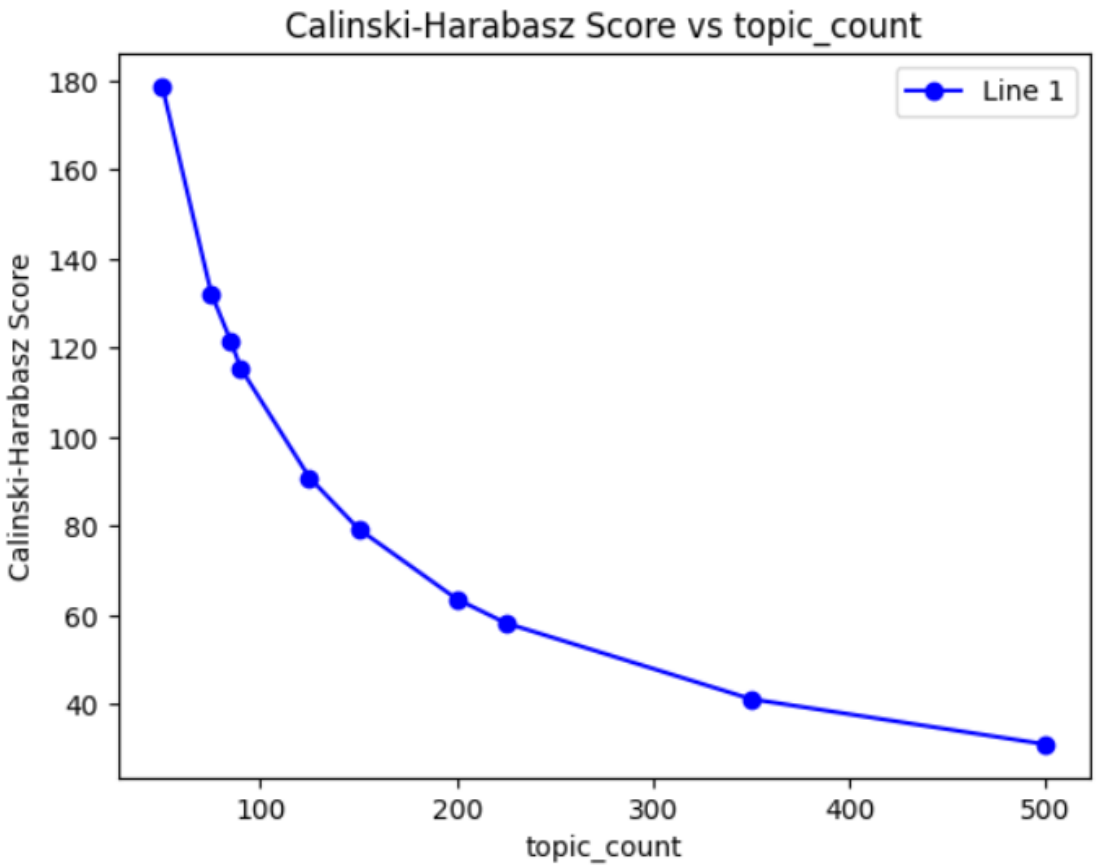


Figure 2: Calinski-Harabasz Score versus Topic Count

B Source Code

The full source code can be found on GitHub:

`https://github.com/cirons2003/document_retrieval_research`[20]

Bibliography

[1] R. Bellman. *Dynamic Programming*. Princeton University Press, 1957.

[2] I. Beltagy, M. E. Peters, and A. Cohan. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*, 2020.

[3] J. Brutlag. Speed matters for google web search. https://services.google.com/fh/files/blogs/google$_l atency_w hitepaper.pdf$, 2009.

[4] D. D. Chamberlin and R. F. Boyce. Sequel: A structured english query language. *Proceedings of the 1974 ACM SIGFIDET Workshop on Data Description, Access and Control*, pages 249–264, 1974.

[5] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979.

[6] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[7] N. Garg, B. Mitra, Q. Lin, and et al. Document ranking with a pre-trained sequence-to-sequence model. *arXiv preprint arXiv:2003.06713*, 2020. https://arxiv.org/abs/2003.06713.

[8] J. A. Hartigan and M. A. Wong. Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[9] J. Healy. Hdbscan, fast density based clustering, the how and the why. https://www.youtube.com/watch?v=6z7GQewK-Ks, 2018.

[10] L. Jin, B. C. Ooi, A. K. H. Tung, and S. Wang. A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality. *IEEE Transactions on Knowledge and Data Engineering*, 18(3):302–313, 2006.

[11] J. Johnson, M. Douze, and H. Jégou. Faiss: Facebook ai similarity search. https://github.com/facebookresearch/faiss, 2017.

[12] K. S. Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28(1):11–21, 1972.

[13] A. Kandula, E. Bernhardsson, D. Simcha, and et al. Scann: Efficient vector similarity search at scale. *arXiv preprint arXiv:2007.04939*, 2020. https://arxiv.org/abs/2007.04939.

[14] V. Karpukhin, B. Oguz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W. tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020. https://arxiv.org/pdf/2004.04906v2.

[15] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, 1990.

[16] O. Khattab and M. Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over bert. *arXiv preprint arXiv:2004.12832*, 2020.

[17] M. Kleppmann. *Designing Data-Intensive Applications: The big ideas behind reliable, scalable, and maintainable systems*. O'Reilly Media, 2017.

[18] L. McInnes, J. Healy, and S. Astels. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205, 2017.

[19] A. Mvd. Okcupid profiles, 2018. Accessed: 2025-04-02.

[20] OpenAI. Chatgpt (version 4.0), 2025. Assisted with generation of utility functions in code, summarization of academic papers and background concepts, and proofreading of some sections. Available at: https://chat.openai.com.

[21] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 3982–3992, Hong Kong, China, 2019. Association for Computational Linguistics.

[22] N. Reimers and I. Gurevych. Ukplab/sentence-transformers. https://github.com/UKPLab/sentence-transformers, 2020.

[23] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at trec-3. In *NIST Special Publication*, 1995.

[24] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[25] E. Rowell. Big-o algorithm complexity cheat sheet. 2021. Accessed: 2025-04-02.

[26] SciPy Developers. Scipy sparse matrices documentation. https://docs.scipy.org/doc/scipy/reference/sparse.html, 2023.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, pages 5998–6008, 2017.

[28] P. Verma. S2 chunking: A hybrid framework for document segmentation through integrated spatial and semantic analysis. *arXiv preprint*, arXiv:2501.05485, January 2025.

[29] M. Zaheer, G. Guruganesh, K. A. Dubey, J. Ainslie, C. Alberti, S. Ontanon, P. Pham, A. Ravula, Q. Wang, L. Yang, and A. Ahmed. Big bird: Transformers for longer sequences. In *Advances in Neural Information Processing Systems*, volume 33, pages 17283–17297, 2020.

[30] J. Zhao, W. Liu, Y. Liu, S. Cheng, L. Li, H. Zeng, Z. Liu, and M. Sun. Moc: Mixtures of text chunking learners for retrieval-augmented generation system. *arXiv preprint*, arXiv:2503.09600, March 2025.