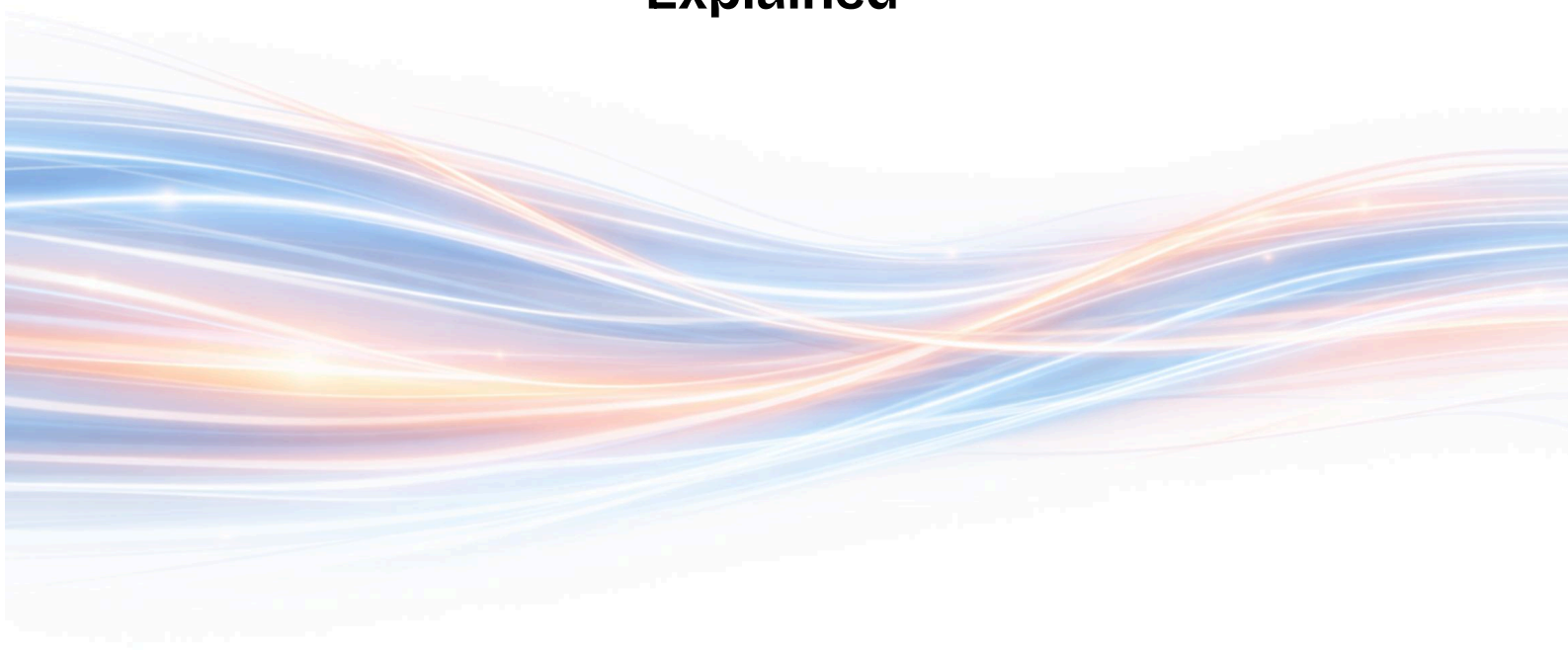
Whitepaper

Multi-Aspect Vector Search Index

Explained



Aspected

Introduction

This paper details a state of the art architecture that enables a more efficient way to search for relevant documents based on different 'aspects' of a document, such as content type, priority or sensitivity labels, timestamps and actual textual data. It aims to elevate existing vector embedding techniques into a truly multi-dimensional retrieval paradigm. By creating a pipeline capable of transforming 'standard' textual data or metadata and computed 'enriched' attributes into separate aspects, we can innovate how we search for documents fundamentally and substantially increase quality. This technique can retrieve documents in a single query, where previously multi stage queries were needed. This leads to a substantial increase in performance also, while decreasing the overall complexity of the system and the consumption of resources such as processing power (CPU time), memory usage and energy required to do these tasks. On top of that, it allows us to perform nearest-neighbor search queries based on holistic relevance scoring that combines both semantic and structural similarity. Previously this was only possible on semantic data, now we can even find relevant documents on imperfect structural matches, enabling "fuzzy" search on the entire representation of a document.

Recent advances in information searching and document retrieval systems have led to the use of vector embeddings to represent the contents of documents and search queries through vectors that exist in a high-dimensional vector space. In vector-based embedding techniques, semantically similar items are represented by vectors that lie closer together within this vector space based on an appropriate distance metric.

When vector embedding techniques are used for semantic document searching, textual data (e.g. words, sentences, paragraphs, or the entire document) is transformed using trained language models, such as Bidirectional Encoder Representations from Transformers (BERT) Devlin et al. (2019) or Generative Pre-trained Transformer (GPT). The textual data is thereby represented using high-dimensional vectors to enable effective matching of queries to documents based on the meaning of the document content rather than on literal matching between keywords in a query and words in the documents.

However, quickly and accurately retrieving the most relevant documents from a large-scale repository remains a challenge.

Problem statement

In typical architectures, vector search is applied to a single embedding per document derived from textual data. Structured metadata, like file type, timestamps, ownership, or a sensitivity label, are indexed separately and applied through pre-query constraints or post-retrieval filtering. This approach often introduces several problems. Firstly, pre-query constraints create empty search spaces, this reduces recall and the ability to return imperfect matches is completely lost. Secondly, post-retrieval filtering has to overshoot the search to be able to find enough matches, which heavily reduces recall again. Plus, in the worst case the algorithm need to loop over the entire dataset to get a desirable number of results.

Furthermore, this architectural separation introduces several run time inefficiencies. Multiple queries or processing stages are often required to combine semantic retrieval with structured filtering, which increases latency. As a result, the overall complexity of the system and the consumption of resources increase, both in terms of computation and maintenance.

Taken together, these limitations highlight the need for more expressive, complete, and complex vector representations that extend beyond typical text embeddings alone.

Muti-Aspect Architecture

Aspects extends the principles of vector search to a broader and more unified representation of documents. Its core contribution lies in generalizing the concept of vector embeddings to encompass all meaningful document properties, referred to as 'aspects'.

An aspect is defined as some form of structured attribute that defines the document as it is. We do not use the term metadata here, as it is generally seen as 'extra' or 'side' information as the name suggests. Nonetheless, we believe that information such as creation dates and content types is just as important a detail for a document as the text itself. As such, Aspects does include, but is not limited to, metadata.

Instead of limiting vectorization to unstructured text, Aspects treats attributes such as content type, creation date, storage context, and enriched semantic annotations as first-class components of document representation. Each aspect is transformed into an aspect vector, with dimensionality and scaling tailored to the real-world behavior of the underlying data. These individual aspect vectors are then fused into a single full aspect embedding, forming a unified vector representation of the document.

In this sense, Aspects does not replace vector search but rather demonstrates how vector search can be elevated into a truly multi-dimensional retrieval paradigm. An interesting effect is that Aspects makes optimizing metadata relevant again. In order to have meaningful vectors, the data that gets transformed into them has to be meaningful also, so both 'standard' metadata and 'enriched' attributes should be appropriately defined and optimized for all documents that are vectorized, such that all documents are well represented in the index.

Traditional Pipeline

The current status-quo vector search architectures can only deal with structured metadata and semantic textual data. Textual data is transformed into a vector embedding and stored in a vector index, while structured data is processed and stored separately, as shown in Fig. 1. However, this means we cannot perform nearest-neighbor search queries on the structured data, as it is not stored in the vector index as an embedding. To search on structured data, multiple queries in the form of post-retrieval filtering or pre-query constraints are necessary. On top of that, pre-query constraints are not possible for all implementations of nearest-neighbor algorithms, making the problem at hand quite complex. An example pipeline for applying post-retrieval filtering is shown in Fig. 2.

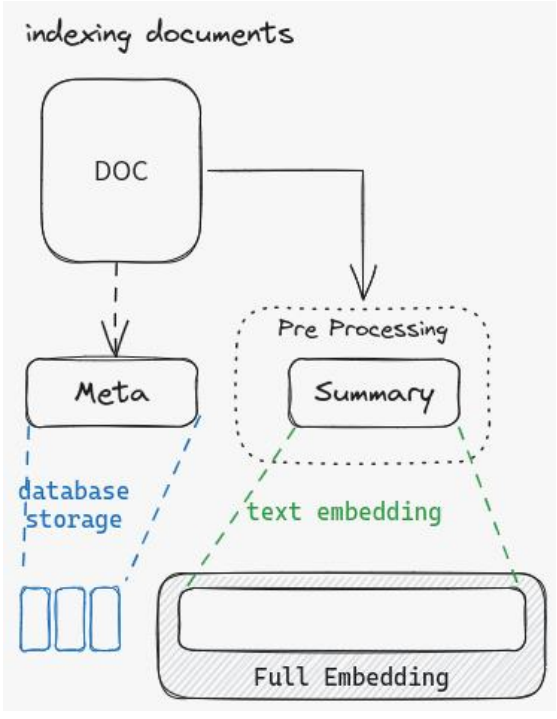


Fig. 1 Traditional Indexing

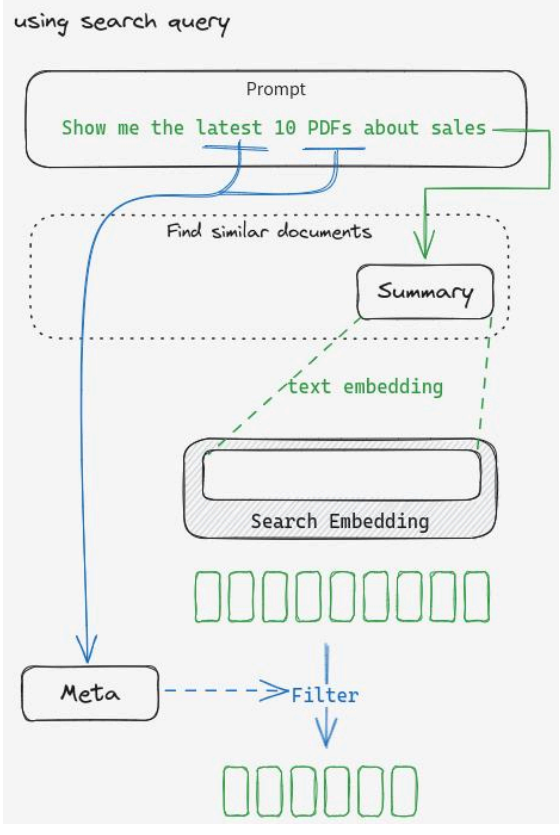


Fig. 2 Traditional Searching

Aspects Pipeline

Aspects introduces generalized vector embeddings, it does not only transform textual data into vector embeddings, but it is also capable of transforming 'standard' metadata and 'enriched' attributes into vector embeddings. Once all the data representing a document has been transformed, all resulting aspect-vectors are fused together into a single full aspect embedding. This indexing process is shown in Fig. 3.

Not only can metadata be included in the vector representation of a document, but enriching attributes can also be generated and transformed into the same vector embedding. This allows complex analytical, generative, or transformative processes to enrich the search space of semantical meaning in which we can search.

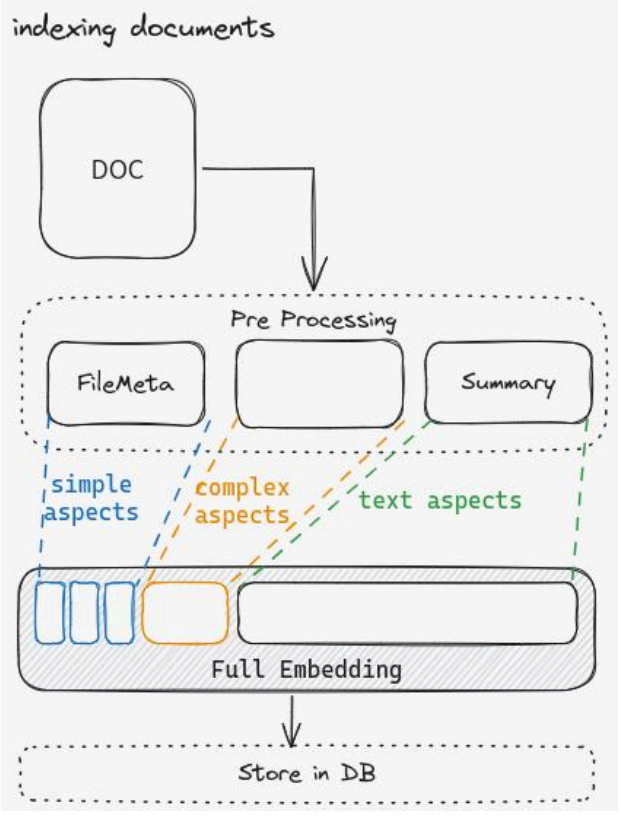


Fig. 3 Aspects Indexing

Sparse search embeddings

The core concept of Aspects is that it treats any attribute, whether it comes from metadata or not, as first-class component and transforms it as part of the full unified vector representation of a document. However, an important detail that needs to be solved is sparse search queries. A large-scale repository of documents has a large potential number of attributes, and it is extremely unlikely that a search query will specify all of these attributes in detail. For Aspects to be useful in the field, it needs to be able to handle those missing details.

This poses two problems. Firstly, not all data structures are capable of searching for sparse search queries, as all dimensions together define the structure in memory. Luckily we can sidestep this problem by picking a data structure that does not organize based on the structure of an embedding itself, but rather on the relationship between different embeddings. A suitable data structure for this is, for example, Hierarchical Navigable Small Worlds Malkov and Yashunin (2016). Secondly, sparse vectors have traditionally been though of as impractical, as any given dimension of the vector embedding cannot just be 'ignored'. All dimensions will contribute to the resulting similarity score between two embeddings, even if a given dimensions contain 0's. This problem becomes very clear when looking at a commonly used distance metric; the cosine distance similarity:

$$sim(\mathbf{q}, \mathbf{d}) = 1 - \frac{\mathbf{q} \cdot \mathbf{d}}{\|\mathbf{q}\| \, \|\mathbf{d}\|}$$

$$= 1 - \frac{\sum_{i=1}^{N} q_i \, d_i}{\sqrt{\sum_{i=1}^{N} q_i^2} \sqrt{\sum_{i=1}^{N} d_i^2}}$$

Where **q** represents the query vector, **d** represents the document vector with which we are calculating **q**'s similarity and *N* is the number of dimensions. Even when one of the dimensions of either vector contains 0's, this same dimension will still contribute to the length of the other vector, influencing the result. Meaning that we cannot entirely 'ignore' a single dimension without altering the document vector **d** as well. We require an additional argument to tell us how to perform this modification.

To remedy this problem, the distance similarity function can be extended to also take a weight vector **w**:

$$sim(\mathbf{q}, \mathbf{d}, \mathbf{w}) = 1 - \frac{\sum_{i=1}^{N} q_i \, d_i \, w_i^2}{\sqrt{\sum_{i=1}^{N} q_i^2 \, w_i^2} \sqrt{\sum_{i=1}^{N} d_i^2 \, w_i^2}}$$

Now, if a single dimension is to be 'ignored', we simply pass a 0 in that dimension of weight vector **w**. This will nullify the lengths of both the query and document vector in that dimension, effectively removing that dimension from the equation altogether.

In most actual use-cases, any dimension of **w** will be set to either 0 or 1. However, any real number value can be used to scale the different aspects as to regulate their influence on the computed similarity score. Both **q** and **d** can also be pre-scaled as to tailor the index to the expected behavior of the underlying data.

Searching modes

The resulting pipeline allows for various ways of searching for documents. A very common use-case for retrieving documents is finding related documents to a known one. The same process outlined above can be applied to produce a vector embedding for the known document. This vector embedding is used for a single query to the Aspects vector index, no additional queries are required. See Fig. 4.

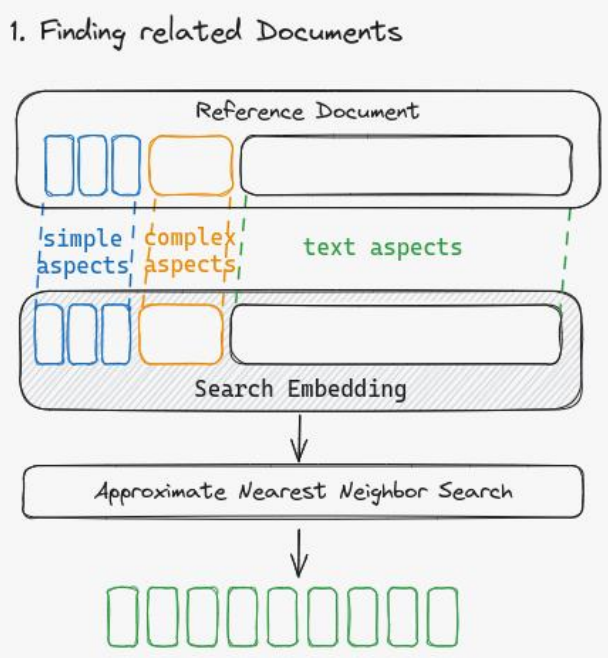


Fig. 4 Aspects related document search

Another possibility is to use a form to collect data. A combination of dropdown menus, input sliders and text inputs could be used to give exact search values for each aspect. These values are then transformed into aspect-vectors again to perform a single query. See Fig. 5.

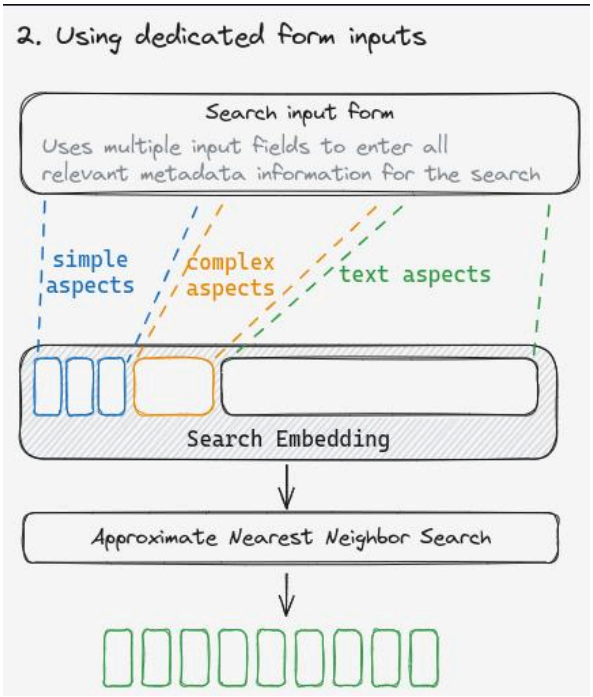


Fig. 5 Aspects search using index forms

Finally natural language queries can be applied to extract aspects from an input prompt. This process is almost equivalent to using an input form, only natural language processing is used to extract the aspect values. See Fig. 6.

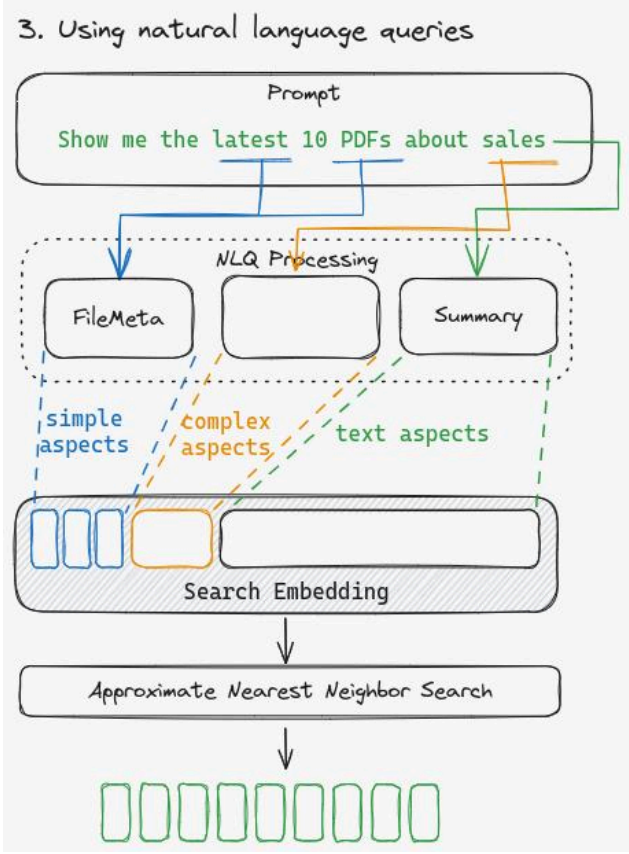


Fig. 6 Aspects search using natural language queries

Aspects Transformers

The process of transforming individual aspects into aspect vectors is intentionally left vague and undefined. Traditionally, we transform textual data using trained language models. This remains a powerful technique to embed semantic data into vector space. However, it is infeasible to use language models for all aspects, including structured data, and simply fuse those together. Language models produce vectors of large dimensions in order to adequately represent semantical meaning. To fuse a potentially large number of such vectors together produces a huge vector embedding, effectively negating any positive effects we hope to gain from our index.

Instead, all structured aspects can be transformed into aspect vectors any way seen fit for the underlying real-world data. This will often mean producing simple hand-made algorithms that transform the data into an aspect vector of very low dimensionality. With proper weighting of all aspects in the full embedding we have seen that even a few dimensions can drastically improve search recall.

Below are three simplified transformers to illustrate the concept. All examples transform the aspect into 1-dimensional aspect vectors. A 1-dimensional aspect vector will essentially divide the rest of the full embedding into groups, separated by a single linear dimension. In our own research this has already proven to be effective for very simple aspect types with a low amount of possible values.

The first example is the most basic; a simple 1-dimensional linear piece of data, such as a timestamp. By simply directly projecting these values onto a single 1-dimensional line starting at -1 and ending in +1 we map all possible values in a one-to-one fashion onto our aspect vector, see Fig. 7.

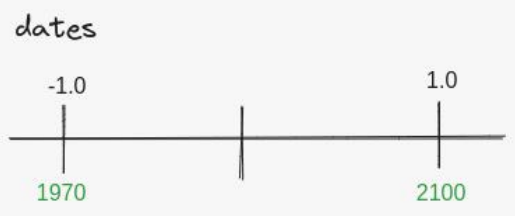


Fig. 7 Data Transformer

A slightly more advanced example would be a non-linear 1-dimensional piece of data, such as a label of a finite number of values, such as the content type of a document. In this case we can project each value onto a range within -1 and +1. The ranges may overlap, as they may share semantical meaning. For example, visual and audio data can be combined and overlap. See Fig. 8. In this example we can already see some complexity rising to the surface, it is unclear whether a document is to the left side of the 'visual' range or to the right side. For these types of data we would often perform some type of analytical process to determine the exact nature of the document.

See Fig. 9 for an example concerning bodies of water, another example where analytical processes can be harnessed to represent non-linear values that are related to each other in same semantical way.

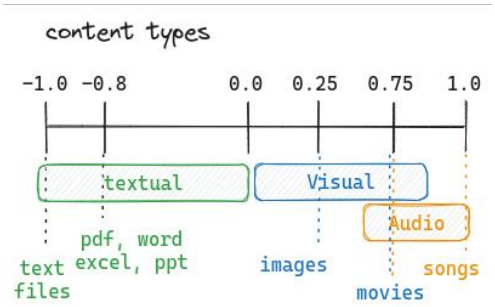


Fig. 8 Content Type Transformer

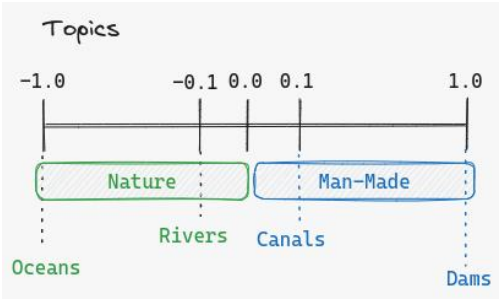


Fig. 9 Topic Transformer

All above examples are simple 1-dimensional examples. However, higher dimensional aspects can represent much more complicated attributes. For example, projecting all data values onto an N-dimensional sphere around the origin in vector space has proven to be much more versatile. This strategy takes advantage of the cosine distance metric, where the angle between different points holds more meaning than the spatial position of each individual point.

References

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

Malkov, Y. A. and Yashunin, D. A. (2016). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs.