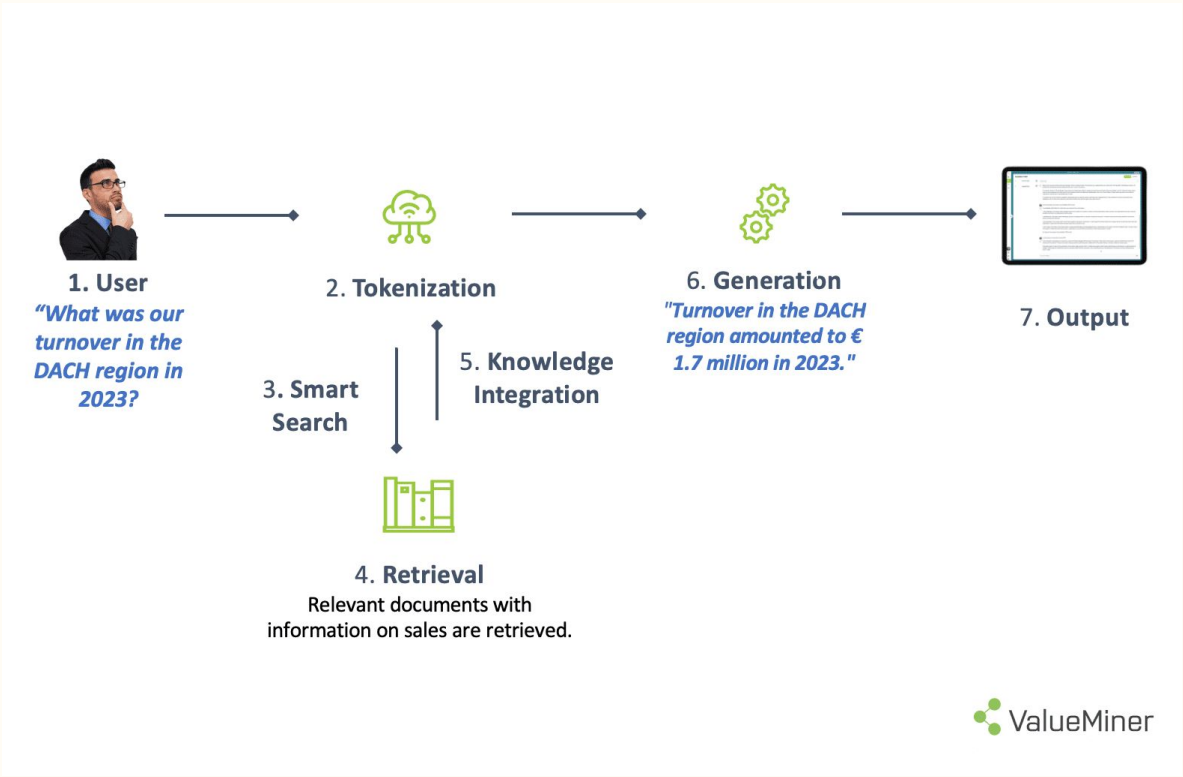
SQUAD + RAG & RAG-Fusion

2025, Jan

Table of Contents

RAG Review


1. User
"What was our turnover in the DACH region in 2023?
2. Tokenization
3. Smart Search
4. Retrieval
Relevant documents with information on sales are retrieved.
5. Knowledge Integration
6. Generation
"Turnover in the DACH region amounted to € 1.7 million in 2023."
7. Output
ValueMiner

Ranking Algorithms

Ranking algorithms used to retrieve information that best “matches” the user’s query:

1. TF-IDF
2. BM25

TF-IDF

TF is the easiest and one of the earliest approaches to retrieval. It states that we need to count the occurrence of the search term in a given document. IDF measures the importance of a term (“dogs” in our case) across the entire corpus.

d1 = cats and dogs are pets.
d2 = cats and dogs are pet animals though I prefer dogs. Dogs obey our commands, can be trained easily and play with us all the time.
d3 = Horses are also pets.

IDF is for the entire corpus

IDF = log (total docs / no of docs containing the term)= log (3 / 2) = 0.176

TF-IDF(“dogs”, d1) = 1 * 0.176 = 0.176

TF-IDF(“dogs”, d2) = 3 * 0.176 = 0.528

TF-IDF(“dogs”, d3) = 0 * 0.176 = 0

Calculating TF-IDF simply boils down to multiplying the two terms TF and IDF. So, the results show that document 2(d2) will be retrieved.

Including Normalization...

It matters how long a sentence is and we cannot ignore that. For example, though “dogs” occurs 3 times in the second document (d2), there are 25 words in it. But the first document(d1) only has 5 words. So let's normalize our TF-IDF by simply dividing it with the document length.

IDF = log (total docs / no of docs containing the term)= log (3 / 2) = 0.176

TF-IDF(“dogs”, d1) = (1 * 0.176) / 5 = 0.0352

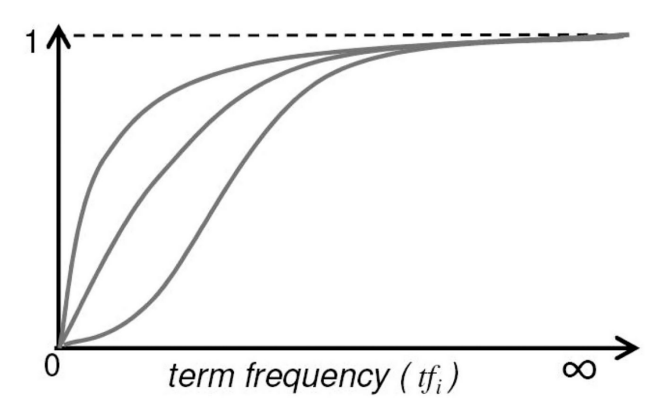
TF-IDF(“dogs”, d2) = (3 * 0.176) / 25 = 0.0212

TF-IDF(“dogs”, d3) = (0 * 0.176) / 4 = 0

Suddenly, our retrieval result return document 1 (d1) instead of document 2(d2) as:
TF-IDF(“dogs”, d1) > TF-IDF(“dogs”, d2).
We can make it more sophisticated by including something called saturation.

Including Saturation...

Saturation is the idea that words should get lesser weightage or value as they occur more and more in a document. Let's consider the second document, “Cats and dogs are pet animals though I prefer dogs. Dogs obey our commands, can be trained easily, and play with us all the time”. What if the document keeps repeating the term, “dogs” over and over again? As the term frequency increases, we will give lesser and lesser weightage to terms as shown in the plot from the paper on BM25.



A plot of saturation of term frequency as it occurs more and more in a document.
Source

BM25

BM25 is a well-regarded retrieval algorithm known for its effectiveness in ranking documents based on their relevance to a given query. It is a robust ranking function employed by search engines to sift through a plethora of documents, ranking them based on their relevance to a user's query. It smartly calculates a score for each document, considering not just the presence but the frequency of query terms, alongside their rarity across the dataset. Introducing BM25 to the retrieval stage of RAG could enhance the model's ability to fetch relevant documents, which in turn could improve the accuracy and relevance of generated responses.

$$BM25 = \sum_{t \in q} \log \left[\frac{N}{df(t)} \right] \cdot \frac{(k_1 + 1) \cdot tf(t,d)}{k_1 \cdot \left[(1 - b) + b \cdot \frac{dl(d)}{dl_{avg}} \right] + tf(t,d)}$$

- k_1, b – parameters
- $dl(d)$ – length of document d
- dl_{avg} – average document length

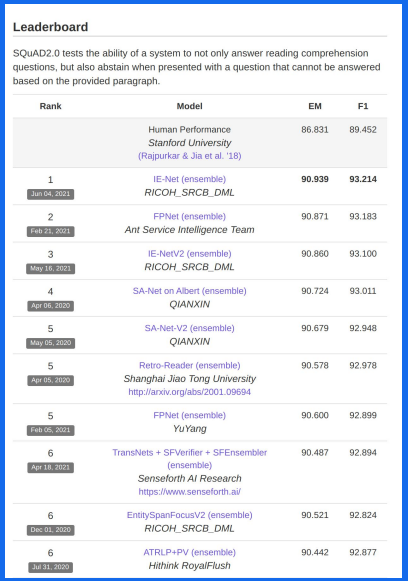
SQuAD (Stanford question answering dataset)

https://rajpurkar.github.io/SQuAD-explorer/

Reading comprehension dataset, consisting of questions posed by crowdworkers on a set of Wikipedia articles, where the answer to every question is a segment of text, or span, from the corresponding reading passage, or the question might be unanswerable

SQuAD2.0 combines the 100,000 questions in SQuAD1.1 with over 50,000 unanswerable questions written adversarially by crowdworkers to look similar to answerable ones. To do well on SQuAD2.0, systems must not only answer questions when possible, but also determine when no answer is supported by the paragraph and abstain from answering.

To evaluate your models, They have also made available the evaluation script they will use for official evaluation, along with a sample prediction file that the script will take as input.



Leaderboard

SQuAD2.0 tests the ability of a system to not only answer reading comprehension questions, but also abstain when presented with a question that cannot be answered based on the provided paragraph.

Rank	Model	EM	F1
	Human Performance *Stanford University* (Rajpurkar & Jia et al. '18)	86.831	89.452
1 Jun 04, 2021	IE-Net (ensemble) *RICOH_SRCB_DML*	**90.939**	**93.214**
2 Feb 21, 2021	FPNet (ensemble) *Ant Service Intelligence Team*	90.871	93.183
3 May 16, 2021	IE-NetV2 (ensemble) *RICOH_SRCB_DML*	90.860	93.100
4 Apr 06, 2020	SA-Net on Albert (ensemble) *QIANXIN*	90.724	93.011
5 May 05, 2020	SA-Net-V2 (ensemble) *QIANXIN*	90.679	92.948
5 Apr 05, 2020	Retro-Reader (ensemble) *Shanghai Jiao Tong University* http://arxiv.org/abs/2001.09694	90.578	92.978
5 Feb 05, 2021	FPNet (ensemble) *YuYang*	90.600	92.899
6 Apr 18, 2021	TransNets + SFVerifier + SFEnsembler (ensemble) *Senseforth AI Research* https://www.senseforth.ai/	90.487	92.894
6 Dec 01, 2020	EntitySpanFocusV2 (ensemble) *RICOH_SRCB_DML*	90.521	92.824
6 Jul 31, 2020	ATRLP+PV (ensemble) *Hithink RoyalFlush*	90.442	92.877

```
version: "v2.0"
data:
  0:
    title: "Normans"
    paragraphs:
      0:
        qas:
          0:
            question: "In what country is Normandy located?"
            id: "56ddde6b9a695914005b9628"
            answers:
              0:
                text: "France"
                answer_start: 159
              1:
                text: "France"
                answer_start: 159
              2:
                text: "France"
                answer_start: 159
              3:
                text: "France"
                answer_start: 159
            is_impossible: false
          1: {…}
          2: {…}
          3: {…}
          4: {…}
          5: {…}
          6: {…}
          7: {…}
          8: {…}
        context: 'The Normans (Norman: Nourmands; French: Normands; Latin: Normanni) were the people who in the 10th and 11th centuries gave their name to Normandy, a region in France. They were descended from Norse ("Norman" comes from "Norseman") raiders and pirates from Denmark, Iceland and Norway who, under their leader Rollo, agreed to swear fealty to King Charles III of West Francia. Through generations of assimilation and mixing with the native Frankish and Roman-Gaulish populations, their descendants would gradually merge with the Carolingian-based cultures of West Francia. The distinct cultural and ethnic identity of the Normans emerged initially in the first half of the 10th century, and it continued to evolve over the succeeding centuries.'
      1: {…}
      2: {…}
      3: {…}
      4: {…}
      5: {…}
      6: {…}
      7: {…}
      8: {…}
      9: {…}
      10: {…}
      11: {…}
      12: {…}
      13: {…}
      14: {…}
      15: {…}
      16: {…}
      17: {…}
```

Limitations of RAG

RAG relies on external knowledge and can produce inaccurate results due to incorrect information.

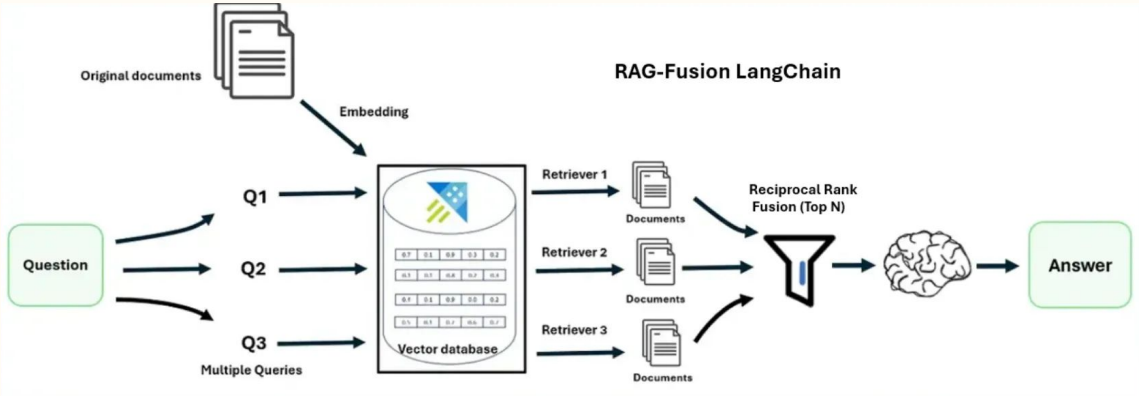
No doubt, humans are not efficient in writing what they require into search engines, and typos, vague queries, and limited vocabulary lead to missing the vast reservoir of information that lies beyond the top results.

The linear paradigm lacks the efficiency to crawl into the depth to understand the nature of human queries. The linear method cannot capture complex user queries, leading to inefficient search results.

RAG Fusion

Inefficiencies in human search and oversimplification of search, lead to less relevant results. RAG Fusion, one can easily overcome the limitations. It overcomes the challenges by generating multiple user queries and ranking the results using strategies like Reciprocal Rank Fusion. The improvised technique bridges the gap between user queries and their intended meaning.

Reciprocal Rank Fusion(RRF), is a data re-ranking technique deployed to combine the results from different queries seamlessly. It aims to organize the search results in a unified ranking, improving the accuracy of relevant information.



RRF (Reciprocal Rank Fusion)

A technique that revolves around combining multiple search results to produce a single, unified ranking. A single query cannot cover all the aspects of user queries, and it might be too narrow to provide comprehensive results; that's why multiple query generation must consider all the different elements and provide a well-curated answer.

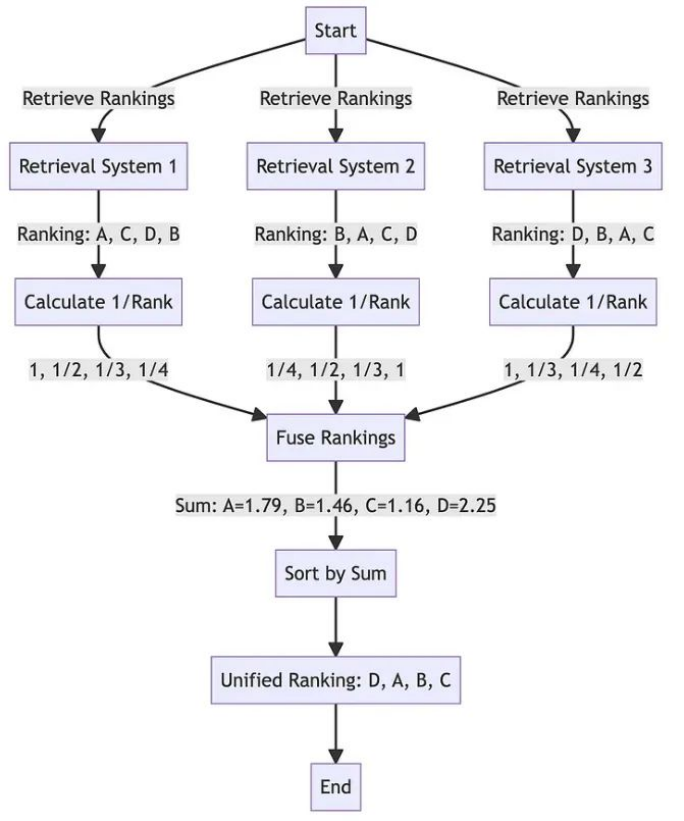
RRF works by combining the ranks of different search queries and increases the chances for all the relevant documents to appear in the final results. Besides, it doesn't rely on absolute scores assigned by search engines but rather on relative ranks, so combining results with different scales or distributions of scores becomes practical.

The RRF algorithm calculates a new score for each document based on its ranks in different lists and sorts them to create a final reranked result.

$$RRFscore(d \in D) = \sum_{r \in R} \frac{1}{k + r(d)},$$

The RRF sorts the documents as per a naive scoring formula. In the above formula, set D represents the given documents to be ranked and a set of rankings R, each a permutation on 1..|D|, and the k is set to 60.

After calculating the fused scores, the function sorts the document into descending order as per the scores and returns the final re-ranked lists.



Probable limitations

- The ability of the RAG Fusion model to reach the depth of query by multiple query generation can provide a detailed answer, more like an overly-explained one.
- The multi-query input and diversified document set can stress the language model's context window, leading to a less coherent output.

Results

Configs:

Evaluation using SQuAD evaluation functions

Baseline RAG:

Training Set = All SQuAD 2.0 training samples

Test Set = 100 samples from training samples

Generative Model = T5-large

Retriever Retrieve top_k = 5

RAG Fusion:

Same as RAG

Query variation Count = 5

Query Generation Method = LLM using t5 (max_length=50, num_beams=10, temperature=0.5) → input: paraphrase: {original_query}

	F1	Exact Match
Baseline RAG	63.55	55
RAG-Fusion	59.23	52