COIL: Revisit Exact Lexical Match in Information Retrieval with Contextualized Inverted List

Luyu Gao, Zhuyun Dai, Jamie Callan
Language Technologies Institute
Carnegie Mellon University
{luyug, zhuyund, callan}@cs.cmu.edu

Abstract

Classical information retrieval systems such as BM25 rely on exact lexical match and carry out search efficiently with inverted list index. Recent neural IR models shifts towards soft semantic matching all query document terms, but they lose the computation efficiency of exact match systems. This paper presents COIL, a contextualized exact match retrieval architecture that brings semantic lexical matching. COIL scoring is based on overlapping query document tokens' contextualized representations. The new architecture stores contextualized token representations in inverted lists, bringing together the efficiency of exact match and the representation power of deep language models. Our experimental results show COIL outperforms classical lexical retrievers and state-of-the-art deep LM retrievers with similar or smaller latency.[1]

1 Introduction

Widely used, bag-of-words (BOW) information retrieval (IR) systems such as BM25 rely on exact lexical match [2] between query and document terms. Recent study in neural IR takes a different approach and compute soft matching between all query and document terms to model complex matching.

The shift to soft matching in neural IR models attempts to address *vocabulary mismatch* problems, that query and the relevant documents use different terms, e.g. cat v.s. kitty, for the same concept (Huang et al., 2013; Guo et al., 2016; Xiong et al., 2017). Later introduction of contextualized representations (Peters et al., 2018) from deep language models (LM) further address *semantic mismatch*, that the same term can refer to different concepts, e.g., bank of river vs. bank in finance. Fine-tuned deep LM rerankers produce token representations based on context and achieve state-of-the-art in text ranking with huge performance leap (Nogueira and Cho, 2019; Dai and Callan, 2019b).

Though the idea of soft matching all tokens is carried through the development of neural IR models, seeing the success brought by deep LMs, we take a step back and ask: how much gain can we get if we introduce contextualized representations back to lexical exact match systems? In other words, can we build a system that still performs exact query-document token matching but compute matching signals with contextualized token representations instead of heuristics? This may seem a constraint on the model, but exact lexical match produce more explainable and controlled patterns than soft matching. It also allows search to focus on only the subset of documents that have overlapping terms with query, which can be done efficiently with inverted list index. Meanwhile, using dense contextualized token representations enables the model to handle semantic mismatch, which has been a long-standing problem in classic lexical systems.

To answer the question, we propose a new lexical matching scheme that uses vector similarities between query-document overlapping term contextualized representations to replace heuristic scoring used in classical systems. We present COntextualized Inverted List (COIL), a new exact lexical match retrieval architecture armed with deep LM representations. COIL processes documents with deep LM offline and produces representations for each document token. The representations are grouped by their surface tokens into inverted lists. At search time, we build representation vectors for query tokens and perform contextualized exact match: use each query token to look up *its own* inverted list and compute vector similarity with document vectors stored in the inverted list as matching scores. COIL enables efficient search with rich-in-semantic matching between query and document.

Our contributions include 1) introduce a novel

[1]Our code is available at `https://github.com/luyug/COIL`.

[2]Exact match up to morphological changes.

retrieval architecture, *contextualized inverted lists* (COIL) that brings semantic matching into lexical IR systems, 2) show matching signals induced from exact lexical match can capture complicated matching patterns, 3) demonstrate COIL significantly outperform classical and deep LM augmented lexical retrievers as well as state-of-the-art dense retrievers on two retrieval tasks.

2 Related Work

Lexical Retriever Classical IR systems rely on exact lexical match retrievers such as Boolean Retrieval, BM25 (Robertson and Walker, 1994) and statistical language models (Lafferty and Zhai, 2001). This type of retrieval model can process queries very quickly by organizing the documents into inverted index, where each distinct term has an inverted list that stores information about documents it appears in. Nowadays, they are still widely used in production systems. However, these retrieval models fall short of matching related terms (vocabulary mismatch) or modeling context of the terms (semantic mismatch). Much early effort was put into improving exact lexical match retrievers, such as matching n-grams (Metzler and Croft, 2005) or expanding queries with terms from related documents (Lavrenko and Croft, 2001). However, these methods still use BOW framework and have limited capability of modeling human languages.

Neural Ranker In order to deal with vocabulary mismatch, neural retrievers that rely on soft matching between numerical text representations are introduced. Early attempts compute similarity between pre-trained word embedding such as word2vec (Mikolov et al., 2013) and GLoVe (Pennington et al., 2014) to produce matching score (Ganguly et al., 2015; Diaz et al., 2016). One more recent approach encodes query and document each into a vector and computes vector similarity (Huang et al., 2013). Later researches realized the limited capacity of a single vector to encode fine-grained information and introduced full interaction models to perform soft matching between all term vectors (Guo et al., 2016; Xiong et al., 2017). In these approaches, scoring is based on learned neural networks and the hugely increased computation cost limited their use to reranking a top candidate list generated by a lexical retriever.

Deep LM Based Ranker and Retriever Deep LM made a huge impact on neural IR. Fine-tuned Transformer (Vaswani et al., 2017) LM BERT (Devlin et al., 2019) achieved state-of-the-art reranking performance for passages and documents (Nogueira and Cho, 2019; Dai and Callan, 2019b). As illustrated in Figure 1a, the common approach is to feed the concatenated query document text through BERT and use BERT's [CLS] output token to produce a relevance score. The deep LM rerankers addressed both vocabulary and semantic mismatch by computing full cross attention between contextualized token representations. Lighter deep LM rankers are developed (MacAvaney et al., 2020; Gao et al., 2020), but their cross attention operations are still too expensive for full-collection retrieval.

Later research therefore resorted to augmenting lexical retrieval with deep LMs by expanding the document surface form to narrow the vocabulary gap, e.g., DocT5Query (Nogueira and Lin, 2019), or altering term weights to emphasize important terms, e.g., DeepCT (Dai and Callan, 2019a). Smartly combining deep LM retriever and reranker can offer additive gain for end performance (Gao et al., 2021a). These retrievers however still suffer from vocabulary and semantic mismatch as traditional lexical retrievers.

Another line of research continues the work on single vector representation and build dense retrievers, as illustrated in Figure 1b. They store document vectors in a dense index and retrieve them through Nearest Neighbours search. Using deep LMs, dense retrievers have achieved promising results on several retrieval tasks (Karpukhin et al., 2020). Later researches show that dense retrieval systems can be further improved by better training (Xiong et al., 2020; Gao et al., 2021b).

Single vector systems have also been extended to multi-vector representation systems. Poly-encoder (Humeau et al., 2020) encodes queries into a set of vectors. Similarly, Me-BERT (Luan et al., 2020) represents documents with a set of vectors. A concurrent work ColBERT (Figure 1c) use multiple vectors to encode both queries and documents (Khattab and Zaharia, 2020). In particular, it represents a documents with all its terms' vectors and a query with an expanded set of term vectors. It then computes all-to-all (Cartesian) soft match between the tokens. ColBERT performs interaction as dot product followed pooling operations, which

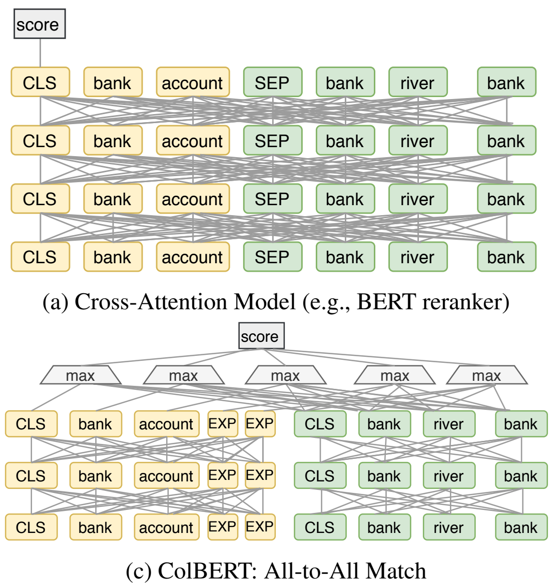
score
CLS
bank
account
SEP
bank
river
bank
score
max
EXP
sum
dot
(a) Cross-Attention Model (e.g., BERT reranker)
(b) Dense Retrievers (e.g., DPR)
(c) ColBERT: All-to-All Match
(d) COIL: Contextualized Exact Match

Figure 1: An illustration of reranking/retrieval mechanisms with deep LM, including our proposed model, COIL.

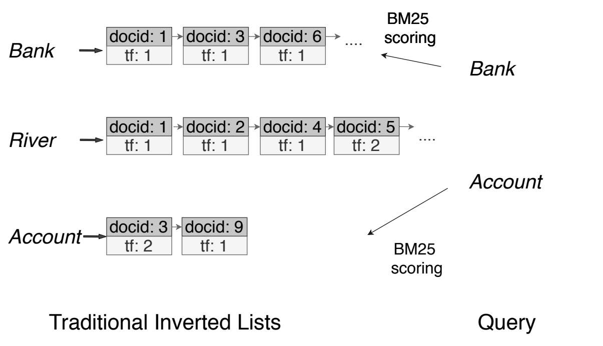

Figure 2: An illustration of traditional inverted lists. The inverted list maps a term to the list of documents where the term occurs. Retriever looks up query terms' inverted lists and scores those documents with stored statistics such as term frequency (tf).

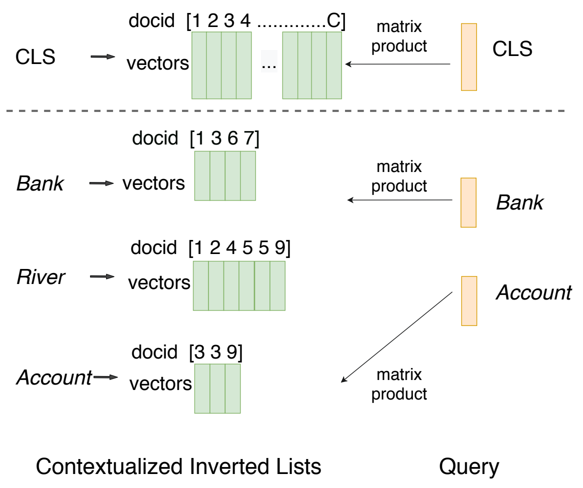

Figure 3: COIL's index and retrieval architecture. COIL-tok relies on the exact token matching (lower). COIL-full includes in addition CLS matching (upper).

allows it to also leverage a dense index to do full corpus retrieval. However, since ColBERT encodes a document with all tokens, it adds another order of magnitude of index complexity to all aforementioned methods: document tokens in the collection need to be stored in a *single* huge index and considered at query time. Consequently, ColBERT is engineering and hardware demanding.

3 Methodologies

In this section, we first provide some preliminaries on exact lexical match systems. Then we discuss COIL's contextualized exact match design and how its search index is organized. We also give a comparison between COIL and other popular retrievers.

3.1 Preliminaries

Classic lexical retrieval system relies on *overlapping* query document terms under morphological generalization like stemming, in other words, *exact lexical match*, to score query document pair. A scoring function is defined as a sum of matched term scores. The scores are usually based on statistics like term frequency (*tf*). Generally, we can write,

$$s = \sum_{t \in q \cap d} \sigma_t(h_q(q,t), h_d(d,t)) \qquad (1)$$

where for each overlapping term t between query q and document d, functions h_q and h_d extract term information and a term scoring function σ_t combines them. A popular example is BM25, which computes,

$$
\begin{aligned}
s_{\text{BM25}} &= \sum_{t \in q \cap d} idf(t) h_q^{\text{BM25}}(q,t) h_d^{\text{BM25}}(d,t) \\
h_q^{\text{BM25}}(q,t) &= \frac{tf_{t,q}(1+k_2)}{tf_{t,q}+k_2} \\
h_d^{\text{BM25}}(d,t) &= \frac{tf_{t,d}(1+k_1)}{tf_{t,d}+k_1(1-b+b\frac{|d|}{\text{avgdl}})}
\end{aligned}
\tag{2}
$$

where $tf_{t,d}$ refers to term frequency of term t in document d, $tf_{t,q}$ refers to the term frequency in query, $idf(t)$ is inverse document frequency, and b, k_1, k_2 are hyper-parameters.

One key advantage of exact lexical match systems lies in efficiency. With summation over exact matches, scoring of each query term only goes to documents that contain matching terms. This can be done efficiently using *inverted list* indexing (Figure 2). The inverted list maps back from a term to a list of documents where the term occurs. To compute Equation 1, the retriever only needs to traverse the subset of documents in query terms' inverted lists instead of going over the entire document collection.

While recent neural IR research mainly focuses on breaking the exact match bottleneck with soft matching of text, we hypothesize that exact match itself can be improved by replacing semantic independent frequency-based scoring with semantic rich scoring. In the rest of this section, we show how to modify the exact lexical match framework with contextualized term representations to build effective and efficient retrieval systems.

3.2 Contextualized Exact Lexical Match

Instead of term frequency, we desire to encode the semantics of terms to facilitate more effective matching. Inspired by recent advancements in deep LM, we encode *both* query and document tokens into contextualized vector representations and carry out matching between exact lexical matched tokens. Figure 1d illustrates the scoring model of COIL.

In this work, we use a Transformer language model[3] as the contextualization function. We encode a query q with the language model (LM) and represent its i-th token by projecting the corresponding output:

$$
\boldsymbol{v}_i^q = \boldsymbol{W}_{tok}\text{LM}(q,i) + \boldsymbol{b}_{tok} \tag{3}
$$

[3]We used the base, uncased variant of BERT.

where $\boldsymbol{W}_{tok}^{n_t \times n_{lm}}$ is a matrix that maps the LM's n_{lm} dimension output into a vector of lower dimension n_t. We down project the vectors as we hypothesize that it suffices to use lower dimension token vectors. We confirm this in section 5. Similarly, we encode a document d's j-th token d_j with:

$$
\boldsymbol{v}_j^d = \boldsymbol{W}_{tok}\text{LM}(d,j) + \boldsymbol{b}_{tok} \tag{4}
$$

We then define the *contextualized exact lexical match scoring function* between query document based on vector similarities between exact matched query document token pairs:

$$
s_{\text{tok}}(q,d) = \sum_{q_i \in q \cap d} \max_{d_j = q_i} (\boldsymbol{v}_i^{q\intercal} \boldsymbol{v}_j^d) \tag{5}
$$

Note that, importantly, the summation goes through only overlapping terms, $q_i \in q \cap d$. For each query token q_i, we finds all *same* tokens d_j in the document, computes their similarity with q_i using the *contextualized* token vectors. The maximum similarities are picked for query token q_i. Max operator is adopted to capture the most important signal (Kim, 2014). This fits in the general lexical match formulation, with h_q giving representation for q_i, h_t giving representations for all $d_j = q_i$, and σ_t compute dot similarities between query vector with document vectors and max pool the scores.

As with classic lexical systems, s_{tok} defined in Equation 5 does not take into account similarities between lexical-different terms, thus faces vocabulary mismatch. Many popular LMs (Devlin et al., 2019; Yang et al., 2019; Liu et al., 2019) use a special CLS token to aggregate sequence representation. We project the CLS vectos with $\boldsymbol{W}_{cls}^{n_c \times n_{lm}}$ to represent the entire query or document,

$$
\begin{aligned}
\boldsymbol{v}_{cls}^q &= \boldsymbol{W}_{cls}\text{LM}(q,\text{CLS}) + \boldsymbol{b}_{cls} \\
\boldsymbol{v}_{cls}^d &= \boldsymbol{W}_{cls}\text{LM}(d,\text{CLS}) + \boldsymbol{b}_{cls}
\end{aligned}
\tag{6}
$$

The similarity between $\boldsymbol{v}_{cls}^q$ and $\boldsymbol{v}_{cls}^d$ provides high-level semantic matching and mitigates the issue of vocabulary mismatch. The full form of COIL is:

$$
s_{\text{full}}(q,d) = s_{\text{tok}}(q,d) + \boldsymbol{v}_{cls}^{q\;\intercal} \boldsymbol{v}_{cls}^d \tag{7}
$$

In the rest of the paper, we refer to systems with CLS matching **COIL-full** and without **COIL-tok**.

COIL's scoring model (Figure 1d) is fully differentiable. Following earlier work (Karpukhin et al., 2020), we train COIL with negative log likelihood defined over query q, a positive document d^+ and a set of negative documents $\{d_1^-, d_2^-, ..d_l^- ..\}$ as loss.

$$\mathcal{L} = -\log \frac{\exp(s(q, d^+))}{\exp(s(q, d^+)) + \sum_l \exp(s(q, d_l^-))} \tag{8}$$

Following Karpukhin et al. (2020), we use in batch negatives and hard negatives generated by BM25. Details are discussed in implementation, section 4.

3.3 Index and Retrieval with COIL

COIL pre-computes the document representations and builds up a search index, which is illustrated in Figure 3. Documents in the collection are encoded offline into token and CLS vectors. Formally, for a unique token t in the vocabulary V, we collect its contextualized vectors from all of its mentions from documents in collection C, building token t's contextualized inverted list:

$$I^t = \{\boldsymbol{v}_j^d \mid d_j = t, d \in C\}, \tag{9}$$

where $\boldsymbol{v}_j^d$ is the BERT-based token encoding defined in Equation 4. We define search index to store inverted lists for all tokens in vocabulary, $\mathbb{I} = \{I^t \mid t \in V\}$. For COIL-full, we also build an index for the CLS token $I^{cls} = \{\boldsymbol{v}_{cls}^d \mid d \in C\}$.

As shown in Figure 3, in this work we implement COIL's by stacking vectors in each inverted list I^t into a matrix $M^{n_t \times |I^k|}$, so that similarity computation that traverses an inverted list and computes vector dot product can be done efficiently as one matrix-vector product with optimized BLAS (Blackford et al., 2002) routines on CPU or GPU. All $\boldsymbol{v}_{cls}^d$ vectors can also be organized in a similar fashion into matrix M_{cls} and queried with matrix product. The matrix implementation here is an exhaustive approach that involves all vectors in an inverted list. As a collection of dense vectors, it is also possible to organize each inverted list as an approximate search index (Johnson et al., 2017; Guo et al., 2019) to further speed up search.

When a query q comes in, we encode every of its token into vectors $\boldsymbol{v}_i^q$. The vectors are sent to the subset of COIL inverted lists that corresponds query tokens $\mathbb{J} = \{I^t \mid t \in q\}$. where the matrix product described above is carried out. This is efficient as $|\mathbb{J}| << |\mathbb{I}|$, having only a small subset of all inverted lists to be involved in search. For COIL-full, we also use encoded CLS vectors $\boldsymbol{v}_{cls}^q$ to query the CLS index to get the CLS matching scores. The scoring over different inverted lists can serve in parallel. The scores are then combined by Equation 5 to rank the documents.

Readers can find detailed illustration figures in the Appendix A, for index building and querying, Figure 4 and Figure 5, respectively.

3.4 Connection to Other Retrievers

Deep LM based Lexical Index Models like DeepCT (Dai and Callan, 2019a, 2020) and DocT5Query (Nogueira and Lin, 2019) alter $tf_{t,d}$ in documents with deep LM BERT or T5. This is similar to a COIL-tok with token dimension $n_t = 1$. A single degree of freedom however measures more of a term *importance* than *semantic agreement*.

Dense Retriever Dense retrievers (Figure 1b) are equivalent to COIL-full's CLS matching. COIL makes up for the lost token-level interactions in dense retriever with exact matching signals.

ColBERT ColBERT (Figure 1c) computes relevance by soft matching *all* query and document term's contextualized vectors.

$$s(q, d) = \sum_{q_i \in [cls;q;exp]} \max_{d_j \in [cls;d]} (\boldsymbol{v}_i^{q\intercal} \boldsymbol{v}_j^d) \tag{10}$$

where interactions happen among query q, document d, cls and set of query expansion tokens exp. The all-to-all match contrasts COIL that only uses exact match. It requires a dense retrieval over all document tokens' representations as opposed to COIL which only considers query's overlapping tokens, and are therefore much more computationally expensive than COIL.

4 Experiment Methodologies

Datasets We experiment with two large scale ad hoc retrieval benchmarks from the TREC 2019 Deep Learning (DL) shared task: MSMARCO passage (8M English passages of average length around 60 tokens) and MSMARCO document (3M English documents of average length around 900 tokens)[4]. For each, we train models with the MSMARCO Train queries, and record results on MSMARCO Dev queries and TREC DL 2019 test queries. We report mainly full-corpus retrieval results but also include the rerank task on MSMARCO Dev queries where we use neural scores to reorder BM25 retrieval results provided by MSMARO organizers. Official metrics include

[4]Both datasets can be downloaded from `https://microsoft.github.io/msmarco/`

MRR@1K and NDCG@10 on test and MRR@10 on MSMARCO Dev. We also report recall for the dev queries following prior work (Dai and Callan, 2019a; Nogueira and Lin, 2019).

Compared Systems Baselines include 1) traditional exact match system BM25, 2) deep LM augmented BM25 systems DeepCT (Dai and Callan, 2019a) and DocT5Query (Nogueira and Lin, 2019), 3) dense retrievers, and 4) soft all-to-all retriever ColBERT. For DeepCT and DocT5Query, we use the rankings provided by the authors. For dense retrievers, we report two dense retrievers trained with BM25 negatives or with mixed BM25 and random negatives, published in Xiong et al. (2020). However since these systems use a robust version of BERT, RoBERTa (Liu et al., 2019) as the LM and train document retriever also on MSMARCO passage set, we in addition reproduce a third dense retriever, that uses the exact same training setup as COIL. All dense retrievers use 768 dimension embedding. For ColBERT, we report its published results (available only on passage collection). BERT reranker is added in the rerank task.

We include 2 COIL systems: 1) COIL-tok, the exact token match only system, and 2) COLL-full, the model with both token match and CLS match.

Implementation We build our models with Pytorch (Paszke et al., 2019) based on huggingface transformers (Wolf et al., 2019). COIL's LM is based on BERT's base variant. COIL systems use token dimension $n_t = 32$ and COIL-full use CLS dimension $n_c = 768$ as default, leading to 110M parameters. We add a Layer Normalization to CLS vector when useful. All models are trained for 5 epochs with AdamW optimizer, a learning rate of 3e-6, 0.1 warm-up ratio, and linear learning rate decay, which takes around 12 hours. Hard negatives are sampled from top 1000 BM25 results. Each query uses 1 positive and 7 hard negatives; each batch uses 8 queries on MSMARCO passage and 4 on MSMARCO document. Documents are truncated to the first 512 tokens to fit in BERT. We conduct validation on randomly selected 512 queries from corresponding train set. Latency numbers are measured on dual Xeon E5-2630 v3 for CPU and RTX 2080 ti for GPU. We implement COIL's inverted lists as matrices as described in subsection 3.3, using NumPy (Harris et al., 2020) on CPU and Pytorch on GPU. We perform a) a set of matrix products to compute token similarities over contextualized inverted lists, b) scatter to map token scores back to documents, and c) sort to rank the documents. Illustration can be found in the appendix, Figure 5.

5 Results

This section studies the effectiveness of COIL and how vector dimension in COIL affects the effectiveness-efficiency tradeoff. We also provide qualitative analysis on contextualized exact match.

5.1 Main Results

Table 1 reports various systems' performance on the MARCO passage collection. COIL-tok exact lexical match only system significantly outperforms all previous lexical retrieval systems. With contextualized term similarities, COIL-tok achieves a MRR of 0.34 compared to BM25's MRR 0.18. DeepCT and DocT5Query, which also use deep LMs like BERT and T5, are able to break the limit of heuristic term frequencies but are still limited by semantic mismatch issues. We see COIL-tok outperforms both systems by a large margin.

COIL-tok also ranks top of the candidate list better than dense retrieves. It prevails in MRR and NDCG while performs on par in recall with the best dense system, indicating that COIL's token level interaction can improve precision. With the CLS matching added, COIL-full gains the ability to handle mismatched vocabulary and enjoys another performance leap, outperforming all dense retrievers.

COIL-full achieves a very narrow performance gap to ColBERT. Recall that ColBERT computes all-to-all soft matches between all token pairs. For retrieval, it needs to consider for each query token *all* mentions of *all* tokens in the collection (MSMARCO passage collection has around 500M token mentions). COIL-full is able to capture matching patterns as effectively with exact match signals from only query tokens' mentions and a single CLS matching to bridge the vocabulary gap.

We observe a similar pattern in the rerank task. COIL-tok is already able to outperform dense retriever and COIL-full further adds up to performance with CLS matching, being on-par with ColBERT. Meanwhile, previous BERT rerankers have little performance advantage over COIL [5]. In practice, we found BERT rerankers to be much more

[5]Close performance between COIL and BERT rerankers is partially due to the bottleneck of BM25 candidates.

Table 1: MSMARCO passage collection results. Results not applicable are denoted '–' and no available 'n.a.'.

	MS MARCO Passage Ranking				
	Dev Rerank	Dev Retrieval		DL2019 Retrieval	
Model	MRR@10	MRR@10	Recall@1K	NDCG@10	MRR@1K
Lexical Retriever					
BM25	–	0.184	0.853	0.506	0.825
DeepCT	–	0.243	0.909	0.572	0.883
DocT5Query	–	0.278	0.945	0.642	0.888
BM25+BERT reranker	0.347	–	–	–	–
Dense Retriever					
Dense (BM25 neg)	n.a.	0.299	0.928	0.600	n.a.
Dense (rand + BM25 neg)	n.a.	0.311	0.952	0.576	n.a.
Dense (our train)	0.312	0.304	0.932	0.635	0.898
ColBERT	0.349	0.360	0.968	n.a.	n.a.
COIL-tok	0.336	0.341	0.949	0.660	0.915
COIL-full	0.348	0.355	0.963	0.704	0.924

Table 2: MSMARCO document collection results. Results not applicable are denoted '–' and no available 'n.a.'.

	MS MARCO Document Ranking				
	Dev Rerank	Dev Retrieval		DL2019 Retrieval	
Model	MRR@10	MRR@10	Recall@1K	NDCG@10	MRR@1K
Lexical Retriever					
BM25	–	0.230	0.886	0.519	0.805
DeepCT	–	0.320	0.942	0.544	0.891
DocT5Query	–	0.288	0.926	0.597	0.837
BM25+BERT reranker	0.383	–	–	–	–
Dense Retriever					
Dense (BM25 neg)	n.a.	0.299	0.928	0.600	n.a.
Dense (rand + BM25 neg)	n.a.	0.311	0.952	0.576	n.a.
Dense (our train)	0.358	0.340	0.883	0.546	0.785
COIL-tok	0.381	0.385	0.952	0.626	0.921
COIL-full	0.388	0.397	0.962	0.636	0.913

expensive, requiring over 2700 ms for reranking compared to around 10ms in the case of COIL.

Table 2 reports the results on MSMARCO document collection. In general, we observe a similar pattern as with the passage case. COIL systems significantly outperform both lexical and dense systems in MRR and NDCG and retain a small advantage measured in recall. The results suggest that COIL can be applicable to longer documents with a consistent advantage in effectiveness.

The results indicate exact lexical match mechanism can be greatly improved with the introduction of contextualized representation in COIL. COIL's token-level match also yields better fine-grained signals than dense retriever's global match signal. COIL-full further combines the lexical signals with dense CLS match, forming a system that can deal with both vocabulary and semantic mismatch, being as effective as all-to-all system.

5.2 Analysis of Dimensionality

The second experiment tests how varying COIL's token dimension n_t and CLS dimension n_c affect model effectiveness and efficiency. We record retrieval performance and latency on MARCO passage collection in Table 3.

In COIL-full systems, reducing CLS dimension from 768 to 128 leads to a small drop in performance on the Dev set, indicating that a full 768 dimension may not be necessary for COIL. Keeping CLS dimension at 128, systems with token dimension 32 and 8 have very small performance difference, suggesting that token-specific semantic consumes much fewer dimensions. Similar pattern in n_t is also observed in COIL-tok ($n_c = 0$).

On the DL2019 queries, we observe that reducing dimension actually achieves better MRR. We believe this is due to a regulatory effect, as the

Table 3: Performance and latency of COIL systems with different representation dimensions. Results not applicable are denoted '–' and no available 'n.a.'. Here n_c denotes COIL CLS dimension and n_t token vector dimension. *: ColBERT use approximate search and quantization. We exclude I/O time from measurements.

		Dev Retrieval		DL2019 Retrieval		Latency/ms	
Model		MRR@10	Recall@1K	NDCG@10	MRR	CPU	GPU
BM25		0.184	0.853	0.506	0.825	36	n.a.
Dense		0.304	0.932	0.635	0.898	293	32
ColBERT		0.360	0.968	n.a.	n.a.	458*	–
COIL							
n_c	n_t						
768	32	0.355	0.963	0.704	0.924	380	41
128	32	0.350	0.953	0.692	0.956	125	23
128	8	0.347	0.956	0.694	0.977	113	21
0	32	0.341	0.949	0.660	0.915	67	18
0	8	0.336	0.940	0.678	0.953	55	16

Table 4: Sample query document pairs with similarity scores produced by COIL. Tokens in examination are colored blue. Numbers in brackets are query-document vector similarities computed with vectors generated by COIL.

Query Token	COIL Contextualized Exact Match Score	Relevance
what is a **cabinet** in govt	**Cabinet [16.28]** (government) A **cabinet [16.75]** is a body of high-ranking state officials, typically consisting of the top leaders of the	+
	Cabinet [7.23] is 20x60 and top is 28x72. .... I had a 2cm granite counter-top installed with a 10 inch overhang on one side and a 14 inch....	-
what is priority **pass**	Priority **Pass [11.61]** is an independent airport lounge access program. A membership provides you with access to their network of over 700	+
	Snoqualmie **Pass [7.98]** is a mountain **pass [6.83]** that carries Interstate 90 through the Cascade Range in the U.S. State of Washington....	-
what **is** njstart	NJSTART **is [1.25]** a self-service online platform that allows vendors to manage forms, certifications, submit proposals, access training	+
	Contract awardees will receive their Blanket P.O. once it **is [-0.10]** converted, and details regarding that process will also be sent...	-

test queries were labeled differently from the MS-MARCO train/dev queries (Craswell et al., 2020).

We also record CPU and GPU search latency in Table 3. Lowering COIL-full's CLS dimension from 768 to 128 gives a big speedup, making COIL faster than DPR system. Further dropping token dimensions provide some extra speedup. The COIL-tok systems run faster than COIL-full, with a latency of the same order of magnitude as the traditional BM25 system. Importantly, lower dimension COIL systems still retain a performance advantage over dense systems while being much faster. We include ColBERT's latency reported in the original paper, which was optimized by approximate search and quantization. All COIL systems have lower latency than ColBERT even though our current implementation does not use those optimization techniques. We however note that approximate search and quantization are applicable to COIL, and leave the study of speeding up COIL to future work.

5.3 Case Study

COIL differs from all previous embedding-based models in that it does not use a single unified embedding space. Instead, for a specific token, COIL learns an embedding space to encode and measure the semantic similarity of the token in different contexts. In this section, we show examples where COIL differentiates different senses of a word under different contexts. In Table 4, we show how the token similarity scores differ across contexts in relevant and irrelevant query document pairs.

The first query looks for "cabinet" in the context of "govt" (abbreviation for "government"). The two documents both include query token "cabinet" but of a different concept. The first one refers to the government cabinet and the second to a case or cupboard. COIL manages to match "cabinet" in the query to "cabinet" in the first document with a much higher score. In the second query, "pass" in both documents refer to the concept of permission. However, through contextualization, COIL captures the variation of the same concept and assigns a higher score to “pass” in the first document.

Stop words like “it”, “a”, and “the” are commonly removed in classic exact match IR systems as they are not informative on their own. In the third query, on the other hand, we observe that COIL is able to differentiate “is” in an explanatory sentence and “is” in a passive form, assigning the first higher score to match query context.

All examples here show that COIL can go beyond matching token surface form and introduce rich context information to estimate matching. Differences in similarity scores across mentions under different contexts demonstrate how COIL systems gain strength over lexical systems.

6 Conclusion and Future Work

Exact lexical match systems have been widely used for decades in classical IR systems and prove to be effective and efficient. In this paper, we point out a critical problem, semantic mismatch, that generally limits all IR systems based on surface token for matching. To fix semantic mismatch, we introduce contextualized exact match to differentiate the same token in different contexts, providing effective semantic-aware token match signals. We further propose contextualized inverted list (COIL) search index which swaps token statistics in inverted lists with contextualized vector representations to perform effective search.

On two large-scale ad hoc retrieval benchmarks, we find COIL substantially improves lexical retrieval and outperforms state-of-the-art dense retrieval systems. These results indicate large headroom of the simple-but-efficient exact lexical match scheme. When the introduction of contextualization handles the issue of semantic mismatch, exact match system gains the capability of modeling complicated matching patterns that were not captured by classical systems.

Vocabulary mismatch in COIL can also be largely mitigated with a high-level CLS vector matching. The full system performs on par with more expensive and complex all-to-all match retrievers. The success of the full system also shows that dense retrieval and COIL’s exact token matching give complementary effects, with COIL making up dense system’s lost token level matching signals and dense solving the vocabulary mismatch probably for COIL.

With our COIL systems showing viable search latency, we believe this paper makes a solid step towards building next-generation index that stores semantics. At the intersection of lexical and neural systems, efficient algorithms proposed for both can push COIL towards real-world systems.

References

S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, Greg Henry, M. Héroux, L. Kaufman, Andrew Lumsdaine, A. Petitet, R. Pozo, K. Remington, and C. Whaley. 2002. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28.

Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M Voorhees. 2020. Overview of the trec 2019 deep learning track. *arXiv preprint arXiv:2003.07820*.

Zhuyun Dai and J. Callan. 2019a. Context-aware sentence/passage term importance estimation for first stage retrieval. *ArXiv*, abs/1910.10687.

Zhuyun Dai and J. Callan. 2020. Context-aware document term weighting for ad-hoc search. *Proceedings of The Web Conference 2020*.

Zhuyun Dai and Jamie Callan. 2019b. Deeper text understanding for IR with contextual neural language modeling. In *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2019, Paris, France, July 21-25, 2019*, pages 985–988. ACM.

J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

Fernando Diaz, Bhaskar Mitra, and Nick Craswell. 2016. Query expansion with locally-trained word embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

Debasis Ganguly, Dwaipayan Roy, Mandar Mitra, and Gareth J. F. Jones. 2015. Word embedding based generalized language model for information retrieval. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Luyu Gao, Zhuyun Dai, and Jamie Callan. 2020. Modularized transfomer-based ranking framework. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*. Association for Computational Linguistics.

Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021a. Rethink training of BERT rerankers in multi-stage retrieval pipeline. In *Advances in Information Retrieval - 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28 - April 1, 2021, Proceedings, Part II*.

Luyu Gao, Zhuyun Dai, Tongfei Chen, Zhen Fan, Benjamin Van Durme, and Jamie Callan. 2021b. Complement lexical retrieval model with semantic residual embeddings. In *Advances in Information Retrieval - 43rd European Conference on IR Research, ECIR 2021, Virtual Event, March 28 - April 1, 2021, Proceedings, Part I*.

J. Guo, Y. Fan, Qingyao Ai, and W. Croft. 2016. A deep relevance matching model for ad-hoc retrieval. *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*.

R. Guo, Philip Y. Sun, E. Lindgren, Quan Geng, David Simcha, Felix Chern, and S. Kumar. 2019. Accelerating large-scale inference with anisotropic vector quantization. *arXiv: Learning*.

Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature*.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*.

Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and J. Weston. 2020. Poly-encoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring. In *ICLR*.

J. Johnson, M. Douze, and H. Jégou. 2017. Billion-scale similarity search with gpus. *ArXiv*, abs/1702.08734.

V. Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Yu Wu, Sergey Edunov, Danqi Chen, and W. Yih. 2020. Dense passage retrieval for open-domain question answering. *ArXiv*, abs/2004.04906.

O. Khattab and M. Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *EMNLP*.

John Lafferty and Chengxiang Zhai. 2001. Document language models, query models, and risk minimization for information retrieval. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Victor Lavrenko and W. Bruce Croft. 2001. Relevance-based language models. In *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Y. Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, M. Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Yi Luan, Jacob Eisenstein, Kristina Toutanova, and M. Collins. 2020. Sparse, dense, and attentional representations for text retrieval. *ArXiv*, abs/2005.00181.

Sean MacAvaney, F. Nardini, R. Perego, N. Tonellotto, Nazli Goharian, and O. Frieder. 2020. Efficient document re-ranking for transformers by precomputing term representations. *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Donald Metzler and W. Bruce Croft. 2005. A markov random field model for term dependencies. In *SIGIR 2005: Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, G. S. Corrado, and J. Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*.

Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage re-ranking with bert. *ArXiv*, abs/1901.04085.

Rodrigo Nogueira and Jimmy Lin. 2019. From doc2query to docttttquery.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc.

Jeffrey Pennington, R. Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. *ArXiv*, abs/1802.05365.

Stephen E Robertson and Steve Walker. 1994. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, L. Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NIPS*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Chenyan Xiong, Zhuyun Dai, J. Callan, Zhiyuan Liu, and R. Power. 2017. End-to-end neural ad-hoc ranking with kernel pooling. *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, J. Liu, P. Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *ArXiv*, abs/2007.00808.

Z. Yang, Zihang Dai, Yiming Yang, J. Carbonell, R. Salakhutdinov, and Quoc V. Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*.

A Appendix

A.1 Index Building Illustration

The following figure demonstrates how the document "apple pie baked ..." is indexed by COIL. The document is first processed by a fine-tuned deep LM to produce for each token a contextualized vector. The vectors of each term "apple" and "juice" are collected to the corresponding inverted list index along with the document id for lookup.

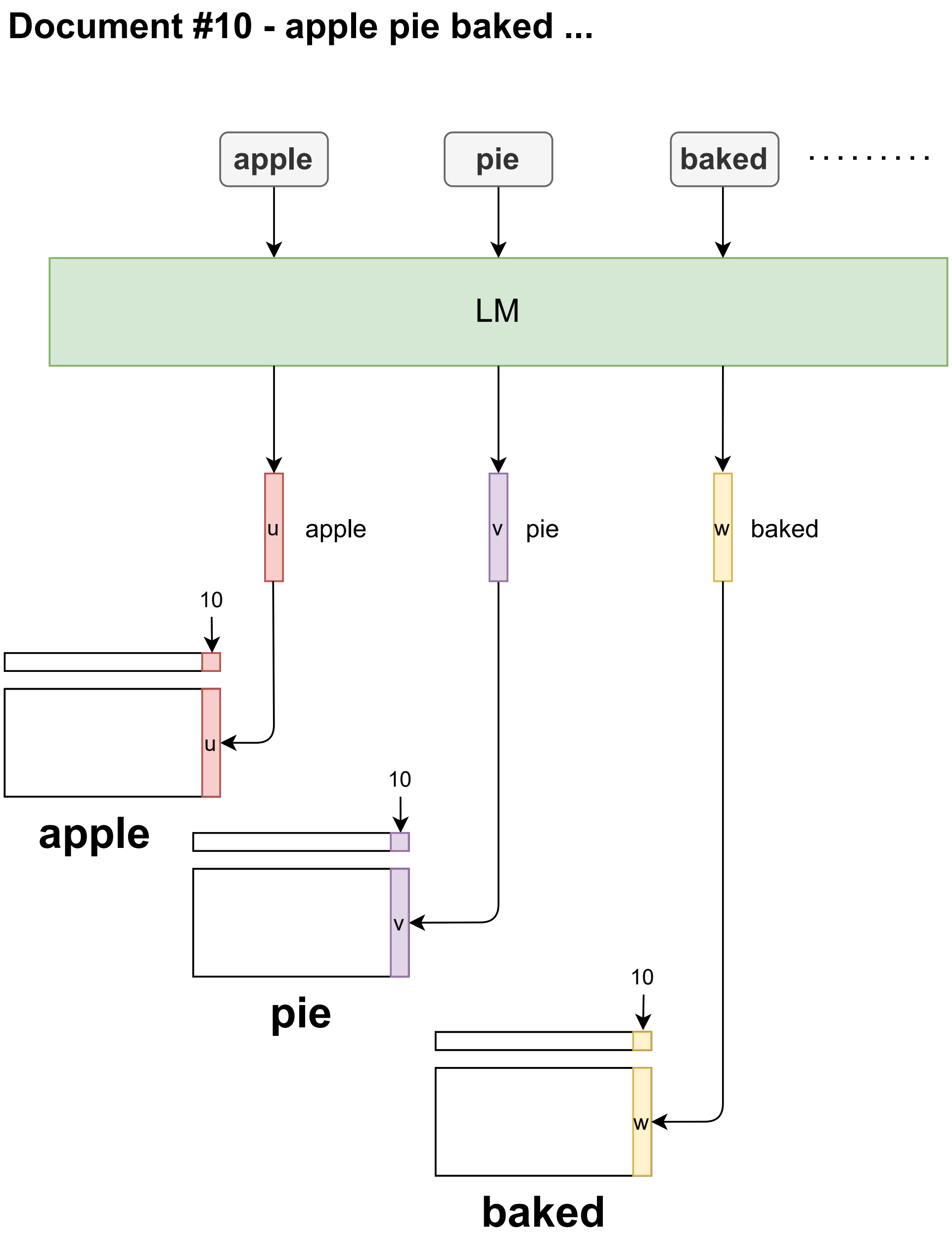

Figure 4: COIL Index Building of document "apple pie baked..."

A.2 Search Illustration

The following figure demonstrates how the query "apple juice" is processed by COIL. Contextualized vectors of each term "apple" and "juice" go to the corresponding inverted list index consisting of a lookup id array and a matrix stacked from document term vectors. For each index, a *matrix vector product* is run to produce an array of scores. Afterwards a *max-scatter* of scores followed by a *sort* produces the final ranking. Note for each index, we show only operations for a subset of vectors (3 vectors) in the index matrix.

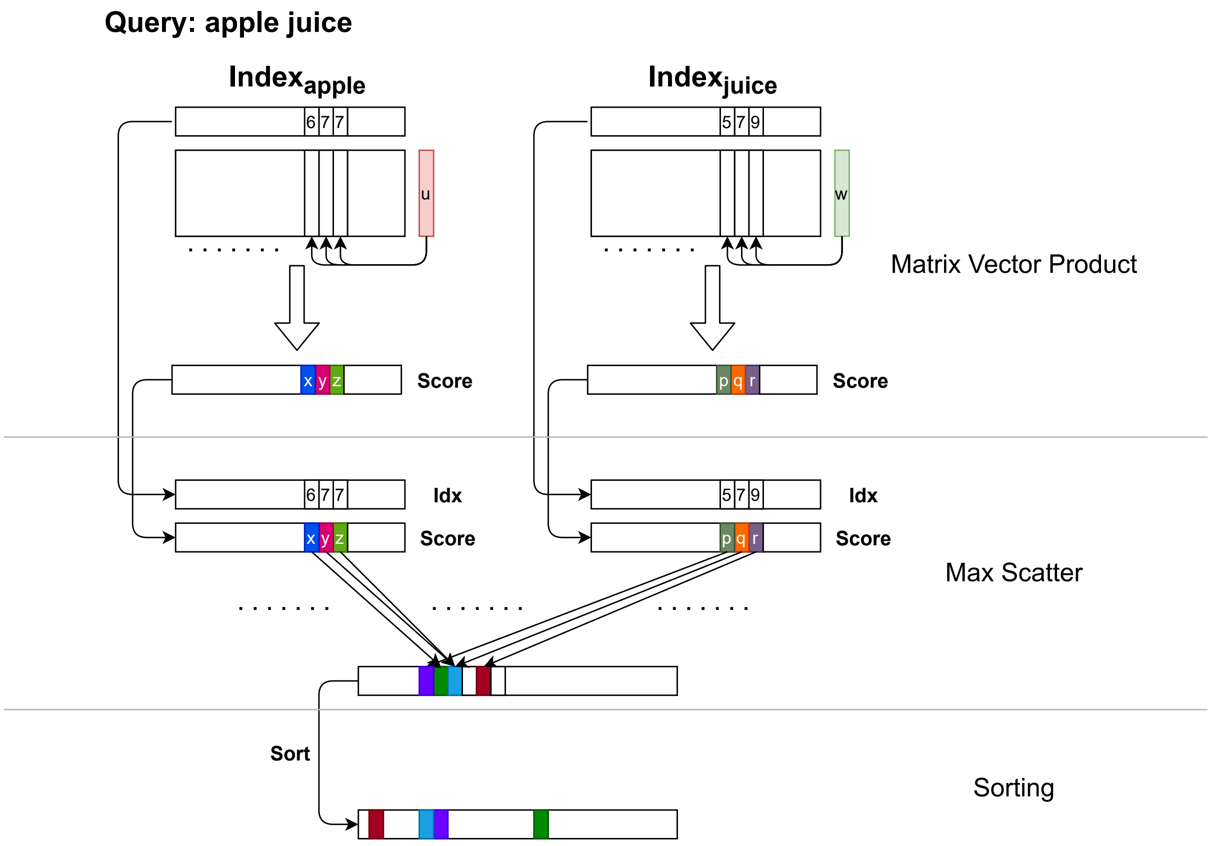

Figure 5: COIL Search of query "apple juice".