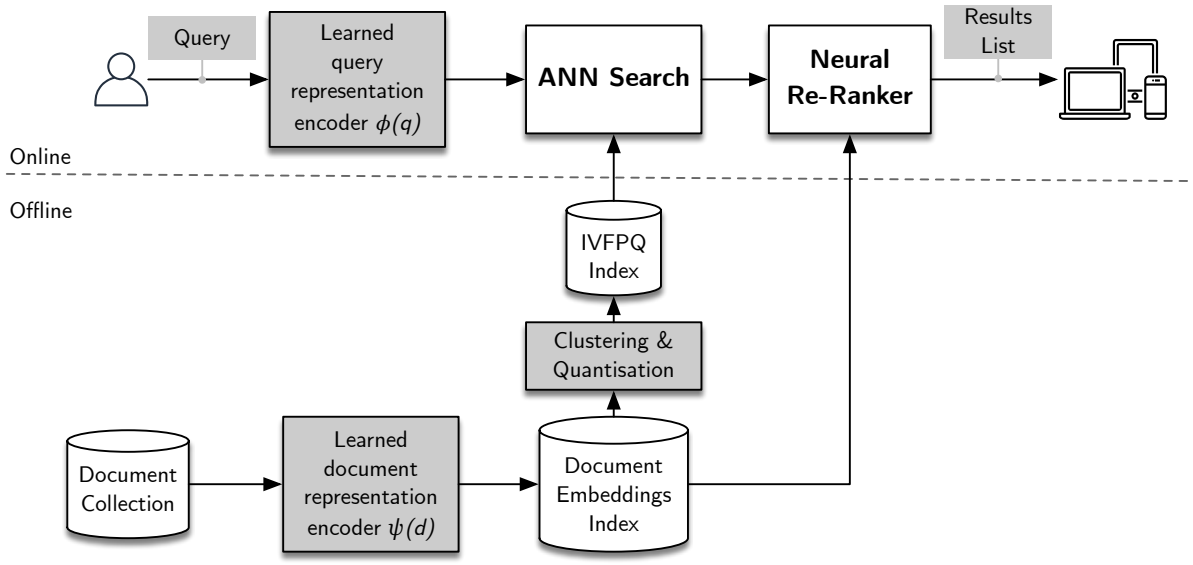

Figure 9: Ranking pipeline architecture for multiple representation systems.

5 Learned Sparse Retrieval

Traditional IR systems are based on sparse representations, inverted indexes and lexical-based relevance scoring functions such as BM25. In industry-scale web search, BM25 is a widely adopted baseline due to its trade-off between effectiveness and efficiency. On the other side, neural IR systems are based on dense representations of queries and documents, that have shown impressive benefits in search effectiveness, but at the cost of query processing times. In recent years, there have been some proposals to incorporate the effectiveness improvements of neural networks into inverted indexes, with their efficient query processing algorithms, through *learned sparse retrieval* approaches. In learned sparse retrieval the transformer architectures are used in different scenarios:

- *document expansion learning*: sequence-to-sequence models are used to modify the actual content of documents, boosting the statistics of the important terms and generating new terms to be included in a document;
- *impact score learning*: the output embeddings of documents provided as input to encoder-only models are further transformed with neural networks to generate a single real value, used to estimate the average relevance contribution of the term in the document;
- *sparse representation learning*: the output embeddings of documents provided as input to encoder-only models are projected with a learned matrix on the collection vocabulary, in order to estimate the relevant terms in a document. These

relevant terms can be part of the documents or not, hence representing another form of document enrichment.

In Sections 5.1, 5.2, and 5.3, we describe the main existing approaches in these scenarios, respectively.

5.1 Document expansion learning

Document expansion techniques address the vocabulary mismatch problem [Zhao 2012]: queries can use terms semantically similar but lexically different from those used in the relevant documents. Traditionally, this problem has been addressed using query expansion techniques, such as relevance feedback [Rocchio 1971] and pseudo relevance feedback [Lavrenko and Croft 2001]. The advances in neural networks and natural language processing have paved the way to different techniques to address the vocabulary mismatch problem by expanding the documents by learning new terms. Doc2Query [Nogueira et al. 2019b] and DocT5Query [Nogueira and Lin 2019] showed for the first time that transformer architectures can be used to expand the documents' content to include new terms or to boost the statistics of existing termw. Both approaches focus on the same task, that is, generating new queries for which a specific document will be relevant. Given a dataset of query and relevant document pairs, Doc2Query fine-tunes a sequence-to-sequence transformer model [Vaswani et al. 2017], while DocT5Query fine-tunes the T5 model [Raffel et al. 2020] by taking as input the relevant document and generating the corresponding query. Then, the fine-tuned model is used to predict new queries using top k random sampling [Fan et al. 2018a] to enrich the document by appending these queries before indexing, as illustrated in Figure 10. Instead of leveraging the encoder-decoder models for sentence generation and fine-tune them on document expansion, a different approach computes the importance of all terms in the vocabulary w.r.t. a given document and selects the most important new terms to enrich the document, leveraging an encoder-only architecture to compute the document embeddings. TILDEv2 [Zhuang and Zuccon 2021b] exploits the BERT model to compute the [CLS] output embedding of a document, and linearly projects it over the whole BERT vocabulary. In doing so, TILDEv2 computes a probability distribution over the vocabulary, i.e., a document language model, and then adds to the document a certain number of new terms, corresponding to those with the highest probabilities. As another way of expanding documents, SparTerm [Bai et al. 2020] computes a document language model for each BERT output token, including [CLS], and sums them up to compute the term importance distribution over the vocabulary for the given document. Finally, a learned *gating mechanism* only keeps a sparse

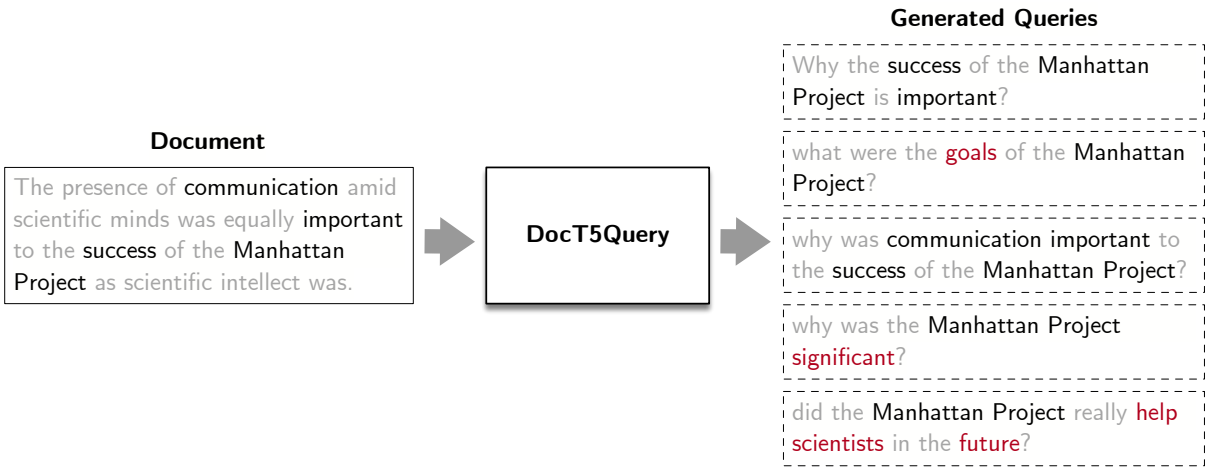

Figure 10: Example of generated queries using DocT5Query. Black terms denote boosted important terms, red terms denote new important terms not present in the original document.

subset of those, to compute the final expanded document contents.

5.2 Impact score learning

Classical inverted indexes store statistical information on term occurrences in documents in posting lists, one per term in the collection. Every posting list stores a posting for each document in which the corresponding term appears in, and the posting contains a document identifier and the in-document term frequency, i.e., a positive integer counting the number of occurrences of the term in the document. When a new query arrives, the posting lists of the terms in the query are processed to compute the top scoring documents, using a classical ranking function, such as BM25, and efficient query processing algorithms [Tonellotto et al. 2018].

The goal of impact score learning is to leverage the document embeddings generated by an encoder-only model to compute a single integer value to be stored in postings, and to be used as a proxy of the relevance of the term in the corresponding posting, i.e., its *term importance*. The simplest way to compute term importance in a document is to project the document embeddings of each term with a neural network into a single-value representation, filtering out negative values with ReLU functions and discarding zeros. To save space, the real values can be further quantised into a 8-bit positive integers. A common problem in impact score learning is the vocabulary to use. Since most encoder-only models use a sub-word tokeniser, the collection vocabulary can be constructed in two different ways: by using the terms produced by the encoder-specific sub-word tokeniser, e.g., by BERT-like tokenisers, or by using the terms produced by

a word tokeniser. These two alternatives have an impact on the final inverted index: in the former case, we have fewer terms, but longer and denser posting lists, while in the latter case, we have more terms, with shorter posting lists and with smaller query processing times [Mallia et al. 2022].

The current impact score learning systems are DeepCT [Dai and Callan 2019b, 2020], DeepImpact [Mallia et al. 2021], TILDEv2 [Zhuang and Zuccon 2021a,b], and UniCOIL [Lin and Ma 2021].

DeepCT [Dai and Callan 2019b, 2020] represents the first example of term importance boosting. DeepCT exploits the contextualised word representations from BERT to learn new in-document term frequencies, to be used with classical ranking functions such as BM25. For each term $w_i \in \mathcal{V}$ in a given document, DeepCT estimates its context-specific *importance* $z_i \in \mathbb{R}$, that is then scaled and rounded as frequency-like integer value tf_i that can be stored in an inverted index. Formally, for each document $d \in \mathcal{D}$, DeepCT projects the ℓ-dimensional representations ψ_i for each input BERT token w_i in the document, with $i = 1, \ldots, |d|$, into a scalar term importance with the learned matrix $W \in \mathbb{R}^{1\times\ell}$:

$$\begin{aligned} [\psi_0, \psi_1, \ldots] &= \mathsf{Encoder}(d) \\ z_i &= W\psi_i \end{aligned} \tag{24}$$

DeepCT is trained with a per-token regression task, trying to predict the importance of the terms. The actual term importance to predict is derived from the document containing the term, or from a training set of query, relevant document pairs. A term appearing in multiple relevant documents and in different queries has a higher importance than a term matching fewer documents, and/or fewer queries. To handle BERT's sub-word tokens, DeepCT uses the importance of the first sub-word token for the entire word, and when a term occurs multiple times in the document, it takes the maximum importance across the multiple occurrences.

DeepImpact [Mallia et al. 2021] proposes for the first time to directly compute an impact score for each unique term in a document, without resorting to classical ranking functions, but simply summing up, at query processing time, the impacts of the query terms appearing in a document to compute its relevance score. For each term $w_i \in \mathcal{V}$ in a given document $d \in \mathcal{D}$, DeepImpact estimates its context-specific *impact* $z_i \in \mathbb{R}$. DeepImpact feeds the encoder-only model with the document sub-word tokens, producing an embedding for each input token. A non-learned *gating layer* Mask removes the embeddings of the sub-word tokens that do not correspond to the first sub-token of the whole word. Then DeepImpact transforms the remaining ℓ-dimensional representations with two feed forward networks with ReLU activations. The first network has

a weight matrix $W_1 \in \mathbb{R}^{\ell \times \ell}$, and the second network has a weight matrix $W_2 \in \mathbb{R}^{1 \times \ell}$:

$$\begin{aligned} [\psi_0, \psi_1, \ldots] &= \mathsf{Encoder}(\mathsf{DocT5Query}(d)) \\ [x_0, x_1, \ldots] &= \mathsf{Mask}(\psi_0, \psi_1, \ldots) \\ y_i &= \mathsf{ReLU}(W_1 x_i) \\ z_i &= \mathsf{ReLU}(W_2 y_i) \end{aligned} \tag{25}$$

The output real numbers z_i, with $i = 1, \ldots, |d|$, one per whole word in the input document, are then linearly quantised into 8-bit integers that can be stored in an inverted index. This produces a single-value score for each unique term in the document, representing its impact. Given a query q, the score of the document d is simply the sum of impacts for the intersection of terms in q and d. DeepImpact is trained with query, relevant document, non-relevant document triples, and, for each triple, two scores for the corresponding two documents are computed. The model is optimized via pairwise cross-entropy loss over the document scores. Moreover, DeepImpact has been the first sparse learned model leveraging at the same time documents expansion learning and impact score learning. In fact, DeepImpact leverages DocT5Query to enrich the document collection before learning the term impact.

TILDEv2 [Zhuang and Zuccon 2021b] computes the terms' impact with an approach similar to DeepImpact. The main differences are (i) the use of a single layer feed forward network with ReLU activations, instead of a two-layer network, to project the document embeddings into a single positive scalar value using a learned matrix $W \in \mathbb{R}^{1 \times \ell}$, (ii) the use of its own document expansion technique, as discussed in Section 5.1, (iii) the use of an index with sub-word terms instead of whole word terms, and (iv) the selection of the highest-valued impact score for a token if that token appears multiple times in a document:

$$\begin{aligned} [\psi_0, \psi_1, \ldots] &= \mathsf{Encoder}(\mathsf{TILDEv2}(d)) \\ z_i &= \mathsf{ReLU}(W \psi_i) \end{aligned} \tag{26}$$

The z_i scores are then summed up, obtaining an accumulated query-document score.

UniCOIL [Lin and Ma 2021] exploits the COIL approach (see Sec. 3), but instead of projecting the query and document embeddings on 8-32 dimensions, it projects them to single-dimension query weights and document weights. In UniCOIL the query and document `[CLS]` embeddings are not used, and the embeddings corresponding to normal query and document tokens are projected into single scalar values $v_1, \ldots, v_{|d|}$ using a learned matrix $W \in \mathbb{R}^{1 \times \ell}$, with ReLU activations on the output term weights of

the base COIL model to force the model to generate non-negative weights.

$$
\begin{aligned}
[\phi_0, \phi_1, \ldots] &= \mathsf{Encoder}(q) \\
[\psi_0, \psi_1, \ldots] &= \mathsf{Encoder}(\mathsf{DocT5Query}(d)) \\
[v_1, v_2, \ldots] &= [W\phi_1, W\phi_2, \ldots] \\
[z_1, z_2, \ldots] &= [W\psi_1, W\psi_2, \ldots] \\
s(q, d) &= \sum_{t_i \in q} \max_{t_j \in d,\, t_j = t_i} v_i z_j
\end{aligned}
\tag{27}
$$

The document weights z_i are then linearly quantised into 8-bit integers, and the final query-document score is computed by summing up the highest valued document impact scores times its query weight v_i, computed at query processing time, as in Eq. (27).

5.3 Sparse representation learning

Instead of independently learning to expand the documents and then learning the impact score of the terms in the expanded documents, sparse representation learning aims at learning both at the same time. At its core, sparse representation learning projects the output embeddings of an encoder-only model into the input vocabulary, compute, for each input term in the document, a language model, i.e., a probability distribution over the whole vocabulary. These term-based language models capture the semantic correlations between the input term and all other terms in the collection, and they can be used to (i) *expand* the input text with highly correlated terms, and (ii) *compress* the input text by removing terms with low probabilities w.r.t. the other terms. Encoder-only models such as BERT already compute term-based language models, as part of their training as masked language models. Formally, given a document d, together with the output embeddings $\psi_{\texttt{[CLS]}}, \psi_1, \ldots, \psi_{|d|}$, an encoder-only model also returns the *masked language heads* $\chi_1, \ldots, \chi_{|d|}$, one for each token in the document, where $\chi_i \in \mathbb{R}^{|\mathcal{V}|}$ for $i = 1, \ldots, |d|$ is an estimation of the importance of each word in the vocabulary implied by the i-th token in the document d. EPIC [MacAvaney et al. 2020a] and SparTerm [Bai et al. 2020] have been the first systems focusing on vocabulary-based expansion and importance estimation, and inspired the SPLADE [Formal et al. 2021] system, on which we focus.

For a given document $d \in \mathcal{D}$, SPLADE computes its per-token masked language heads $\chi_1, \ldots, \chi_{|d|}$ using BERT, filters and sums up these vocabulary-sized vectors into a single vector $\gamma(d) \in \mathbb{R}^{|\mathcal{V}|}$ representing the whole document, and then uses this vector to

represent the document itself, together with the term importance scores:

$$
\begin{aligned}
[\chi_1, \ldots, \chi_{|d|}] &= \mathsf{Encoder}(d) \\
\gamma(d) &= \sum_{i=1}^{|d|} \log\left(1 + \mathsf{ReLU}(\chi_i)\right)
\end{aligned}
\tag{28}
$$

The logarithm and ReLU functions in Eq. (28) are computed element-wise; the logarithm prevents some terms with large values from dominating, and the ReLU function deals with the negative components of $\gamma(d)$.

The document representation γ potentially contains all terms in the vocabulary, even if the logarithm and ReLU functions in Eq. (28) can zero out some of its components. To learn to "sparsify" the document representations, Formal et al. [2021] leverage the FLOPS regulariser $\mathcal{L}_{\mathsf{FLOPS}}$ [Paria et al. 2020]. As part of the SPLADE loss function used during training, the FLOPS loss is computed as the sum, across the terms in the vocabulary, of the squared probability p_w^2 that a term w has a non-zero weight in a document. Minimising the FLOPS loss coincides with minimising the non-zero weights in a document, i.e., maximising the number of zero weights in a document. The square operation helps in reducing high term weights more than low term weights. The probability that a term $w \in \mathcal{V}$ has a non-zero weight in a document d is proportional to the average weight of that term $\gamma_t(d)$ estimated through the whole collection. To make the computation feasible, the average is computed on a batch b of documents during training, considered as a representative sample of the whole collection:

$$
\mathcal{L}_{\mathsf{FLOPS}} = \sum_{t \in V} p_t^2 = \sum_{t \in V} \left(\frac{1}{|b|} \sum_{d \in b} \gamma_t(d) \right)^2 \tag{29}
$$

SPLADE does not limit expansion to documents only. Indeed, Eq. (28) can be applied to a query q as well, to compute the corresponding vector $\gamma(q) \in \mathbb{R}^{|\mathcal{V}|}$. However, this query expansion must be carried out at query processing time; to reduce the latency, the expanded query should be far more sparse than a document. To enforce this different behaviour, Formal et al. [2021] adopt two distinct FLOPS regularisers for documents and queries, both as in Eq. 29.

6 Conclusions

This overview aimed to provide the foundations of neural IR approaches for ad-hoc ranking, focusing on the core concepts and the most commonly adopted neural architecture in current research. In particular, in Section 1 we provided a background on