Introducing Neural Bag of Whole-Words with ColBERTer: Contextualized Late Interactions using Enhanced Reduction

Sebastian Hofstätter
TU Wien

Omar Khattab
Stanford University

Sophia Althammer
TU Wien

Mete Sertkan
TU Wien

Allan Hanbury
TU Wien

ABSTRACT
Recent progress in neural information retrieval has demonstrated large gains in quality, while often sacrificing efficiency and interpretability compared to classical approaches. We propose ColBERTer, a neural retrieval model using contextualized late interaction (ColBERT) with enhanced reduction. Along the effectiveness Pareto frontier, ColBERTer dramatically lowers ColBERT’s storage requirements while simultaneously improving the interpretability of its token-matching scores. To this end, ColBERTer fuses single-vector retrieval, multi-vector refinement, and optional lexical matching components into one model. For its multi-vector component, ColBERTer reduces the number of stored vectors by learning unique whole-word representations and learning to identify and remove word representations that are not essential to effective scoring. We employ an explicit multi-task, multi-stage training to facilitate very small vector dimensions. Results on the MS MARCO and TREC-DL collection show that ColBERTer reduces the storage footprint by up to 2.5×, while maintaining effectiveness. With just one dimension per token in its smallest setting, ColBERTer achieves index storage parity with the plaintext size, with very strong effectiveness results. Finally, we demonstrate ColBERTer’s robustness on seven high-quality out-of-domain collections, yielding statistically significant gains over traditional retrieval baselines.

CCS CONCEPTS
- Information systems → Learning to rank;

KEYWORDS
Neural Ranking; Dense-Sparse Hybrid Retrieval

ACM Reference Format:
Sebastian Hofstätter, Omar Khattab, Sophia Althammer, Mete Sertkan, and Allan Hanbury. 2022. Introducing Neural Bag of Whole-Words with ColBERTer: Contextualized Late Interactions using Enhanced Reduction. In Proceedings of the 31st ACM Int’l Conference on Information and Knowledge Management (CIKM ’22), Oct. 17–21, 2022, Atlanta, GA, USA. ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3511808.3557367

1 INTRODUCTION
Traditional retrieval systems have long relied on bag-of-words representations to search text collections. This has led to mature architectures, in which compact inverted indexes enable fast top-k retrieval strategies, while also exhibiting interpretable behavior, where retrieval scores can directly be attributed to contributions from individual terms. Despite these qualities, recent progress in Information Retrieval (IR) has firmly demonstrated that pre-trained language models can considerably boost effectiveness over classical approaches. This progress has raises questions about how to control the computational cost and how to ensure interpretability of these neural models. This has sparked an unprecedented tension in IR between achieving the best retrieval quality, maintaining low computational costs, and prioritizing interpretable modeling.

For practical applications, IR architectures are confined to strict cost constraints around query latency and space footprint. While disk space might be affordable, keeping large pre-computed representations in memory—as often needed for low query latency—increases hardware costs considerably. For multi-vector models like ColBERT [22], space consumption is determined by a multiplication of three variables: 1) the number of vectors per document; 2) the number of dimensions per vector; 3) the number of bytes per dimension. This work is motivated by the observation that reducing any of these three variables directly reduces the storage requirement proportionally and yet different choices carry different impact on effectiveness. Well-studied low hanging fruits for good tradeoffs include reducing the number of dimensions and reducing the number of bytes with quantization [12, 19, 25, 34]. Reducing the number of vectors offers a rich design space around model architecture and retrieval strategy.

Besides efficiency, the accelerating adoption of machine learning coincides with indications that future regulatory environments...
will require deployed models to provide transparent and reliably interpretable output to their users. This need for interpretability is especially pronounced in IR, where the ranking models are demanded to be fair and transparent [6]. Despite this, the two largest classes of neural models at the moment—namely, cross-encoders and single-vector bi-encoders—rely on opaque aggregations that conceal the contributions of query and document terms on retrieval scores.

This paper presents a novel end-to-end retrieval model called ColBERTer. ColBERTer extends the popular ColBERT model with effective enhanced reduction approaches. These reductions increase the level of interpretability and reduce the storage and latency cost greatly, while maintaining the quality of retrieval.

ColBERTer fuses a single-vector retrieval and multi-vector refinement model into one with explicit multi-task training. Next, ColBERTer introduces neural Bag of Whole-Words (BOW\(^2\)) representations for increasing interpretability and reducing the number of stored vectors in the ranking process. The BOW\(^2\) consist of the aggregation of all subword token representations contained in a unique whole word. To further reduce the number of vectors, ColBERTer learns to remove BOW\(^2\) representations with simplified contextualized stopwords (CS) [17]. And to reduce the dimensional-ity of the token vectors down to one, our methods employ an Exact Matching (EM) component that aligns representations across only lexical matches from the query and document, a model variant we call Uni-ColBERTer following the nomenclature of Lin and Ma [26].

Figure 1 illustrates ColBERTer’s BOW\(^2\) representation and how we can display whole-word scores to the user in a keyword view. By aggregating all subwords to whole words, the whole-word scores of this complex medical-domain query illustrate ColBERTer’s interpretability capabilities, without cherry picking examples that only contain words that are fully part of BERT’s vocabulary.

The ColBERTer architecture enables various indexing and retrieval scenarios. Building on recent work [12, 26], we provide a holistic categorization and ablation study of five possible usage scenarios of ColBERTer encoded sequences: sparse token retrieval, dense single vector retrieval, as well as refining either one of the retrieval sources and a full hybrid mode. Specifically, we study:

**RQ1** Which aggregation and training regime works best for combined retrieval and refinement capabilities of ColBERTer?

We find that multi-task learning with two weighted loss functions for retrieval and refinement and a learned score aggregation of both consistently outperforms fixed score aggregation. We investigate jointly training aggregation, BOW\(^2\), and contextualized stopwords with a weighted multi-task loss. We find that tuning the weights improves the tradeoff between removed vectors and retrieval quality, but that the results are robust to small hyperparameter changes.

Following our definition of dense and sparse combinations, we study various deployment scenarios and answer:

**RQ2** What is ColBERTer’s best indexing and refinement strategy?

Interestingly, we find that a full hybrid retrieval deployment is unnecessary, and only results in very modest and not significant gains compared to a sparse or dense index with passage refinement of the other component. While a dense index produces higher recall than a sparse one, the effect on the top 10 results becomes negligible after refinement, especially on TREC-DL. This novel result could lead to less complexity in deployment, as only one index is required. Practitioners could choose to keep a sparse index, if they already made significant investments or choose only a dense approximate nearest neighbor index for more predictable query latency. Both sparse and dense encodings of ColBERTer can be optimized with common indexing improvements.

With our hyperparameters fixed, we aim to understand the quality effect of reducing storage factors along 2 axes of ColBERTer:

**RQ3** How do different configurations of dimensionality and vector count affect the retrieval quality of ColBERTer?

We study the effect of BOW\(^2\), CS, and EM reductions on across dimensions (32, 16, 8, and 1) and find that, while retrieval quality is reduced with each dimension reduction, the delta is small. Furthermore, we observe that BOW\(^2\) and CS reductions result – on every dimension setting – in a Pareto improvement over simply reducing the number of dimensions.

While we want to emphasize that it becomes increasingly hard to contrast neural retrieval architectures – due to the diversity surrounding training procedures – and make conclusive statements about “SOTA” – due to evaluation uncertainty – we still compare ColBERTer to related approaches:

**RQ4** How does the fully optimized ColBERTer system compare to other end-to-end retrieval approaches?

We find that ColBERTer improves effectiveness compared to related approaches, especially for systems with low storage footprint. Uni-ColBERTer especially outperforms previous single-dimension token encoding approaches, while offering improved transparency with score mappings to whole words.

To evaluate the robustness of ColBERTer we test it on seven high-quality and diverse collections from different domains. We use a meta-analysis [45] that reveals whether statistical significant gains are achieved over multiple collections. We investigate:

**RQ5** How robust is ColBERTer when applied out of domain?

We find that ColBERTer with token embeddings of 32 or Uni-ColBERTer with 1 dimension both show an overall significantly higher retrieval effectiveness compared to BM25, with not a single collection worse than BM25. Compared to a TAS-Balanced trained dense retriever [16] ColBERTer is not statistically significantly worse on any single collection. While we observe an overall positive effect it is not statistically significant within a 95% confidence interval. This robust analysis tries to not overestimate the benefits of ColBERTer, while at the same time giving us more confidence in the results. We publish our code, trained models, and documentation at: github.com/sebastian-hofstaetter/colberter

## 2 BACKGROUND

This section empirically motivates storing unique whole-word representations, reviews the single-vector BERT\(_{DOT}\) and multi-vector ColBERT architectures, and describes other related approaches.

---

1 Such as a recent 2021 proposal by the EU Commission on AI regulation, see: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52021PC0206 (Art. 13)
2.1 Tokenization

Many modern neural IR models use a BERT [9] variant to contextualize sequences and are thus locked into a specific tokenization scheme. The BERT tokenizer first splits full text on whitespace and punctuation characters and then uses the WordPiece algorithm [44] to split words to sub-word tokens in a reduced vocabulary. Taggregating unique-stemmed whole-words only stores from 59% to 36% of the original sub-word units of BERT in our used collections. Related multi-vector methods, such as ColBERT or (Uni)COIL, generally save all BERT tokens, while our BOW^2 aggregation (§3.2) saves only stemmed unique whole-words.

2.2 BERT^DOT and ColBERT Architectures

BERT^DOT matches a single vector of the query with a single vector of a passage, produced by independent BERT computations [30, 32, 55]. ColBERT [22] delays the interactions between query and document to after the BERT computation. For more information we refer the reader to Hofstätter et al. [15].

2.3 Related Work

Vector Reduction. Previous neural IR work on reducing the number of vectors produce fixed sizes across all passages. Lassane et al. [23] prune ColBERT representations to either 50 or 10 vectors by sorting tokens by Inverse Document Frequency (IDF) or attention scores from BERT. Zhou and Devlin [57] extend ColBERT with temporal pooling, sliding a window over the passage to create a vector per step with a fixed target count. Luan et al. [33] represent each passage with a fixed number of embeddings of the CLS token and the first m token of the passage, and compute relevance as the maximum score of the embeddings. Humeau et al. [18] compute a fixed number of vectors per query, and aggregate them by softmax attention against document vectors. Lee et al. [24] learn phrase (multi-word) representations for QA collections. This reduces the vector count, but it depends on the availability of exact answer spans in passages and is therefore not universally applicable in IR. Tonellotto and Macdonald [47] prune the embeddings of the query terms but the document embeddings.

In summary, unlike related vector reduction techniques we: 1) reduce a dynamic number of vectors per passage; 2) keep a mapping between human-readable tokens and vectors, allowing scoring information to be used in the user interface; 3) learn the full pruning process end-to-end without term-based supervision.

Vector Compression. Ma et al. [34] study various methods to reduce the dimension of dense retrieval vectors. Unlike our study, they find that learned dimension reduction performs poorly. Also for single vector retrieval Zhan et al. [56] optimize product quantization as part of the training. Recently, Santhanam et al. [43] study residual compression of all saved vector tokens as part of the ColBERT end-to-end retrieval setting. There are concurrent efforts revisiting lexical matching with learned sparse representations [10, 12, 26] or learned passage impacts [37], which employ the efficiency of exact lexical matches. Different to our work, they focus on reducing the number of dimensions of the learned embeddings without reducing the number of stored tokens. Many of these approaches can be considered complementary to our proposed methods, and future work should evaluate how well these methods compose to achieve even larger compression rates.

3 ColBERTer: ENHANCED REDUCTION

ColBERT with enhanced reduction, or ColBERTer, combines the encoding architectures of BERT^DOT and ColBERT, while extremely reducing the token storage and latency requirements along the effectiveness Pareto frontier. Our enhancements maintain model transparency, creating a concrete mapping of scoring sources and human-readable whole-words.

ColBERTer independently encodes the query and the document using a transformer encoder like BERT, producing token-level representation similar to ColBERT:

\[
\tilde{q}_{1:m} = \text{BERT}(\text{CLS} ; q_{1:m} ; \text{SEP})
\]

\[
\tilde{p}_{1:n} = \text{BERT}(\text{CLS} ; p_{1:n} ; \text{SEP})
\]

To maximize transparency, we do not apply the query augmentation mechanism of Khattab and Zaharia [22] (see §2.2), which appends MASK tokens to the query with the goal of implicit – and thus potentially opaque – query expansion.

3.1 2-Way Dimension Reduction

Given the transformer encoder output, ColBERTer uses linear layers to reduce the dimensionality of the output vectors in two ways: 1) we use the linear layer \(W_{CLS}\) to control the dimension of the first CLS-token representation (e.g. 128 dimensions):

\[
{q}_{CLS} = \tilde{q}_{1} \cdot W_{CLS}
\]

\[
{p}_{CLS} = \tilde{p}_{1} \cdot W_{CLS}
\]

and 2) the layer \(W_{i}\) projects the remaining tokens down to the token embedding dimension (usually smaller, e.g. 32):

\[
{q}_{1:m} = \tilde{q}_{2:m+1} \cdot W_{i}
\]

\[
{p}_{1:n} = \tilde{p}_{2:n+1} \cdot W_{i}
\]

This 2-way reduction combined with our novel training workflow (§4.1) serves to reduce our space footprint compared to ColBERT and at the same time provides more expressive encodings than a single vector BERT^DOT model. Furthermore, it enables a multitude of potential dense and sparse retrieval workflows (§4.2).

3.2 BOW^2: Bag of Unique Whole-Words

Given the token representations (\(\tilde{q}_{1:m}\) and \(\tilde{p}_{1:n}\)), ColBERTer applies its novel key transformation: BOW^2 to the sequence of vectors. Whereas ColBERT and COIL maintain one vector for each BERT token, including tokens corresponding to sub-words in the BERT vocabulary, we create a single representation for each unique whole word. This serves to further reduce the storage overhead of our model by reducing the number of tokens, while preserving an explicit mapping of score parts to human understandable words.

During tokenization we build a mapping between each sub-word token and corresponding unique whole word (as defined by a simple split on punctuation and whitespace characters). The words can also be transformed through classical IR techniques such as stemming. Then, inside the model we aggregate whole word representations for each whole word \(w\) in passage \(p\) by computing the mean of the embeddings of \(w\’s\) constituent sub-words \(\tilde{p}_{i}\). We get the set of unique whole-word representation of the passage \(p\):

\[
\hat{p}_{1:n} = \frac{1}{|\{\hat{p}_{i} \mid w \in \text{BOW}^2(p)\}|} \sum_{\tilde{p}_{i} \in w} \tilde{p}_{i} \quad \forall w \in \text{BOW}^2(p)
\]
3.3 Simplified Contextualized Stopwords

To further reduce the number of passage tokens to store, we adopt a simplified version of Hofstätter et al. [17]’s contextualized stopwords (CS), which was first introduced for the TK-Sparse model. CS learns a removal gate of tokens solely based on their context-dependent vector representations. We simplify the original implementation of CS and adapt the removal process to fit into the encoding phase of the ColBERTer model.

Every whole-word passage vector \( \hat{p}_j \) is transformed by a linear layer (with weights \( W_s \) and bias \( b_s \)), followed by a ReLU activation, to compute a single-dimensional stopword removal gate \( r_j \):

\[
    r_j = \text{ReLU}(\hat{p}_j W_s + b_s)
\]

(5)

The original implementation [17] masks scores after TK’s kernel-activation, meaning the non-zero gates have to be saved as well, which increases the system’s complexity. In contrast, we directly apply the gate to the representation vectors. In particular, we drop every representation where the gate \( r_j = 0 \), and otherwise scale the magnitude of the remaining representations using their gate scores:

\[
    \hat{p}_j = \hat{p}_j * r_j
\]

(6)

This fully differentiable approach allows us to learn the stopword gate during training and remove all nullified vectors at indexing time, as they do not contribute to document scores. Applying the stopword gate directly to the representation vector allows us to observe much more stable training than the authors of TK-Sparse observed – we do not need to adapt the training procedure with special mechanisms to keep the model from collapsing. Following Hofstätter et al. [17] we train the removal gate with a regularization loss, forcing the stopword removal gate to become active during training (§4.1).

3.4 Matching & Score Aggregation

After we complete the independent encoding of query and passage sequences, we need to match and score them. ColBERTer creates two scores, one for the CLS vector and one for the token vectors. The CLS score is a dot product of the two CLS vectors:

\[
    s_{\text{CLS}} = \hat{q}_{\text{CLS}} \cdot \hat{p}_{\text{CLS}}
\]

(7)

The score follows the scoring regime of ColBERT, with a match matrix of word-by-word dot product and max-pooling the document word dimension followed by a sum over all query words:

\[
    s_{\text{token}} = \sum_{j=1}^{\hat{m}} \max_{i=1..\hat{n}} \hat{q}_j^T \hat{p}_i
\]

(8)

The final score of a query-passage pair is computed with a learned aggregation of the two score components:

\[
    s_{\text{ColBERTer}} = \sigma(y) * s_{\text{CLS}} + (1 - \sigma(y)) * s_{\text{token}}
\]

(9)

where \( \sigma \) is the sigmoid function, and \( y \) is a trainable scalar parameter. For ablations, \( \sigma(y) \) can be set to a fixed number, such as 0.5. While the learned weighting factor may seem superfluous, as the upstream linear layers could already learn to change the magnitudes of the two components, we show in §6.1 that the explicit weighting is crucial to the effectiveness of both components.

3.5 Uni-ColBERTer: Extreme Reduction with Lexical Matching

While ColBERTer considerably reduces the dimension of the representations already, we found in pilot studies that for an embedding dimension of 8 or lower the full match matrix is detrimental to the effectiveness. Lin and Ma [26] showed that a token score model can be effectively reduced to one dimension in UniCOIL. This reduces the token representations to scalar weights, necessitating an alternative mechanism to match query tokens with “similar” document tokens.

To fit the same reduction we need to apply more techniques to our ColBERTer architecture to create Uni-ColBERTer with single dimensional whole word vectors. While we now occupy the same bytes per vector, our vector reduction techniques make Uni-ColBERTer 2.5 times smaller than UniCOIL (on MSMARCO).

To reduce the token encoding to 1 dimension we apply a second linear layer after the contextualized stopword component:

\[
    \hat{q}_{1,m+2} = \hat{q}_{1,n} * W_u
    \hat{p}_{1,m+2} = \hat{p}_{1,n} * W_u
\]

(10)

Furthermore, we need to apply a lexical match bias, following COIL’ s, to only match identical words with each other. This creates engineering challenge: we do not build a global vocabulary with ids of whole-words during training or inference as doing so would make it difficult to saturate modern GPUs, requiring multiple synchronized CPU processes (4-10 depending on the system) that prepare the input with tokenization, data transformation, and subsequent tensor batching of sequences. To keep track of a global vocabulary, these CPU processes would need to synchronize with every token. This is very challenging at best in python multiprocessing while keeping the necessary speed to fully use even a single GPU.

To overcome this problem, we propose approximate lexical interactions by creating an n-bit hash \( H \) from every whole-word without accounting for potential collisions and applying a mask of equal hashes to the match matrix. Depending on the selection of bits to keep this introduces different numbers of collisions. Depending on the collection size one can adjust the number of bits to save from the hash. With the hashed global id of whole words we can adjust the match matrix of whole-words for low dimension token models as follows:

\[
    s_{\text{token}} = \sum_{i=1}^{\hat{m}} \max_{i=1..\hat{n}} \hat{q}_j^T \hat{p}_i
\]

(11)

\( ^2 \)On MSMARCO we found that the first 32 bits of sha256 produce very few collisions (803 collisions out of 1.6 million hashes).
Introducing Neural Bag of Whole-Words with ColBERTer

In practice, we implement this procedure by masking the full match matrix, so that the operation works on batched tensors. Besides allowing us reduce the token dimensionality to one, the lexical matching component of Uni-ColBERTer enables the sparse indexing of tokens in an inverted index, following UniCOIL.

4 MODEL LIFECYCLE

In this section we describe how we train our ColBERTer architecture and how we can deploy the trained model into a retrieval system.

4.1 Training Workflow

We train our ColBERTer model with triples of one query, and two passages where one is more relevant than the other. To incorporate the degree of relevance, as provided by a teacher model we use the Margin-MSE loss [15], formalized as follows:

\[
L_{\text{Margin-MSE}}(M_t) = \text{MSE}(M_t^+ - M_t^-, M_t^+ - M_t^-)
\]  

(12)

Where a teacher model \( M_t \) provides a teacher signal for our student model \( M_s \) (in our case ColBERTer’s output parts). From the outside ColBERTer looks and acts like a single model, however it is in essence a multi-task model: aggregating sequences into a single vector, representing individual words, and actively removing uninformative words. Therefore, we need to train these three components in a balanced form, with a combined loss function:

\[
L = \alpha_b \ast L_b + \alpha_{\text{CLS}} \ast L_{\text{CLS}} + \alpha_{\text{CS}} \ast L_{\text{CS}}
\]

(13)

where \( \alpha \)'s are hyperparameters governing the weighting of the individual losses, which explain in the following. The combined loss for both sub-scores \( L_b \) uses MarginMSE supervision on the final score:

\[
L_b = L_{\text{MarginMSE}}(\tilde{M}_{\text{ColBERTer}})
\]

(14)

In pilot studies and shown in §6.1 we observed that training ColBERTer only with a combined loss strongly reduces the effectiveness of the CLS vector alone. To overcome this issue and be able to use single vector retrieval we define \( L_{\text{CLS}} \) as:

\[
L_{\text{CLS}} = L_{\text{MarginMSE}}(\tilde{M}_{\text{CLS}})
\]

(15)

Finally, to actually force the model to learn sparsity in the removal gate vector \( r \) of the contextualized stopword component, we follow Hofstätter et al. [17] and add an \( L_{\text{CS}} \) loss of the L1-norm of the positive & negative \( r \):

\[
L_{\text{CS}} = ||r^+||_1 + ||r^-||_1
\]

(16)

This introduces some tension in training: the sparsity loss needs to move as many entries to close to zero, while the token loss as part of \( L_b \) needs non-zeros to determine relevance matches. To reduce volatility, we train the enhanced reduction components one after another. We start with a ColBERT checkpoint, followed by the 2-way dimensionality reduction, BOW\(^2\) and CS, and finally for Uni-ColBERTer we apply another round of reduction.

4.2 Indexing and Query Workflow

Once we have trained our ColBERTer model we need to decide how to deploy it into a wider retrieval workflow. ColBERTer’s passage encoding can be fully pre-computed in an offline setting, which allows for low latency query-time retrieval.

Previous works, such as COIL [12] or ColBERT [22] have already established many of the potential workflows. We aim to give a holistic overview of the possible usage scenarios, including ablation studies to select the best method with the lowest complexity. We give a schematic overview over ColBERTer’s retrieval workflows in Figure 2. We assume that all passages have been encoded and stored accessibly by their id. Each of the two storage categories can be transformed into an index structure for fast retrieval: the CLS index uses an (approximate) nearest neighbor index, while the BOW\(^2\) index could use either a dense nearest neighbor index, or a classic inverted index (with activated exact matching component).

Figure 2 shows how we can index both scoring components of ColBERTer and then use the id-based storages to fill in missing scores for passages retrieved only by one index. A similar workflow has been explored by Lin and Lin [28] and Gao et al. [12]. Figure 2 \( 1 \) & \( 3 \) utilize only one retrieval index and fill up the missing scores from the complementary id-based storage. This approach works vice-versa for dense or sparse indices, and represents a clear complexity and additional index storage reduction, at the potential of lower recall. This is akin to a two stage retrieve and re-rank pipeline [15, 16, 29], but such pipeline have been mostly studied with a separate model per stage (which requires larger indexing resources than our single model). Figure 2 \( 2 \) & \( 4 \) represent ablation studies that only rely on one or the other index while disregarding the other scoring part.

Different workflows may considerably affect complexity, storage, and effectiveness. We thus always indicate the type of query workflow used (numbers given in Figure 2) in our results section and conduct an ablation study in §6.1.

5 EXPERIMENT DESIGN

Our main training and inference dependencies are PyTorch [38], HuggingFace Transformers [54], and the nearest neighbor search library Faiss [20]. For training we utilize TAS-Balanced [16] retrieved negatives with BERT-based teacher ensemble scores [15].

5.1 Passage Collection & Query Sets

For training and in-domain evaluation we use the MSMARCO-Passage (V1) collection [3] with the sparsely-judged MSMARCO-DEV query set of 6,980 queries (used in the leaderboard) as well as the densely-judged 97 query set of combined TREC-DL ’19 [7] and ’20 [8]. For TREC graded relevance (0 = non relevant to 3 = perfect), we use the recommended binarization point of 2 for
Table 1: Analysis of different score aggregation and training methods for ColBERTer (2-way dim reduction only; CLS dim: 128, token dim: 32; Workflow Θ) in terms of retrieval effectiveness. We compare refining full-retrieval results from ColBERTer’s CLS vector (Own) and a TAS-Balanced retriever (TAS) with different multi-task loss weights \(\alpha_b\) and \(\alpha_{CLS}\).

| Train Loss | TREC-DL’19+20 | MSMARCO DEV |
|------------|----------------|-------------|
| | nDCG@10 | R@1K | MRR@10 | R@1K |
| Fixed Score Aggregation | Own TAS | Own TAS | Own TAS | Own TAS |
| 1 | 1 | 0 | \(\alpha_{CLS} = 0\) | \(\alpha_b = 0\) | \(.861\) | \(.386\) | \(.336\) | \(.957\) | \(.978\) | \(.387\) | \(.960\) | \(.386\) | \(.962\) | \(.730\) | \(.805\) | \(.565\) | \(.987\) | \(.861\) |
| Learned Score Aggregation | Own TAS | Own TAS | Own TAS | Own TAS |
| 2 | 1 | 0.1 | \(.726\) | \(.728\) | \(.861\) | \(.384\) | \(.952\) | \(.978\) |
| 3 | 1 | 0.2 | \(.728\) | \(.731\) | \(.861\) | \(.384\) | \(.952\) | \(.978\) |
| 4 | 1 | 0.5 | \(.734\) | \(.734\) | \(.861\) | \(.386\) | \(.956\) | \(.978\) |
| 5 | 1 | 1 | \(.730\) | \(.730\) | \(.861\) | \(.381\) | \(.956\) | \(.978\) |

Table 2: Analysis of the bag of whole-words (BOW\(^2\)) and contextualized stopword training of ColBERTer (CLS dim: 128, token dim: 32; Workflow Θ) using different multi-task loss parameters.

| Train Loss | BOW\(^2\) Vectors | D\(\text{L}'19\)+20 | DEV |
|------------|-----------------|-----------------|-----|
| | nDCG@10 | R@1K | MRR@10 | R@1K |
| BOW\(^2\) only | 1 | 0.5 | 0 | 43.2 | 0\% | \(.731\) | \(.815\) | \(.387\) | \(.963\) |
| 2 | 1 | 0.1 | 0 | 43.2 | 0\% | \(.736\) | \(.806\) | \(.387\) | \(.960\) |
| BOW\(^2\) + Contextualized Stopwords | 1 | 0.5 | 1 | 29.1 | 33\% | \(.731\) | \(.811\) | \(.382\) | \(.965\) |
| 4 | 1 | 0.1 | 1 | 27.8 | 36\% | \(.729\) | \(.802\) | \(.385\) | \(.960\) |
| 5 | 1 | 0.75 | 0.9 | 30.9 | 29\% | \(.730\) | \(.805\) | \(.387\) | \(.961\) |
| 6 | 1 | 0.5 | 0.5 | 36.7 | 15\% | \(.725\) | \(.806\) | \(.387\) | \(.962\) |

To isolate the CLS retrieval performance for workflow Θ (dense CLS retrieval, followed by BOW\(^2\) storage refinement) we compare different training and aggregation strategies with ColBERTer’s CLS retrieval vs. re-ranking the candidate set retrieved by a standalone TAS-Balanced retriever in Table 1. Using COIL’s aggregation and training approach (by fixing \(\sigma(y) = 0.5\) in Eq. 9 and setting \(\alpha_{CLS} = 0\)) we observe in line 1 that the CLS retrieval component fails substantially, compared to utilizing TAS-B. We postulate that this happens, as the token refinement component is more capable in determining relevance and therefore it dominates the changes in gradients, which minimizes the standalone capabilities of CLS retrieval. Now, with our proposed multi-task and learned score aggregation (lines 2-5) we observe much better CLS retrieval performance. While it still lacks a bit behind TAS-B in recall, these deficiencies do not manifest itself after refining the token scores for top-10 results in both TREC-DL and MSMARCO DEV. We selected the best performing setting in line 4 for our future experiments.

The next addition in our multi-task framework is the learned removal of stopwords. This adds a third loss function \(L_{CS}\) that conflicts with the objective of the main \(L_e\) loss. Table 2 shows the tradeoff between retained BOW\(^2\) vectors and effectiveness. In lines 1 & 2 we see ColBERTer without the stopword components, here 43 vectors are saved with unique BOW\(^2\) for MSMARCO (compared to 77 for all subword tokens). In lines 3 to 6 we study different loss weighting combinations with CS. While the ratio of removed stopwords is rather sensitive to the selected parameters, the effectiveness values largely remain constant for lines 4 to 6. Based on the MRR value of the DEV set (with the smallest effectiveness change, but still 29% removed vectors) we select configuration 5 going forward, although we stress that our approach would also work well with the other settings, and cherry picking parameters is not needed. This setting reduces the number of vectors and thus footprint by a factor of 2.5 compared to ColBERT, while keeping the same top-10 effectiveness (comparing Table 2 line 5 vs. Table 1 line 1 (TAS-B re-ranked)).

Future work could use a conservative loss setting (such as line 6) that does not force a lot of the word removal gates to become zero (so as to not take away capacity from the loss surface for the ranking tasks), followed by the removal words with a non-zero (but still small) threshold during inference.

Following the ablation of training possibilities, we now turn towards the possible usage scenarios, as laid out in §4.2, and answer:

RQ1 Which aggregation and training regime works best for combined retrieval and refinement capabilities of ColBERTer?
Table 3: Analysis of the retrieval quality for different query-time retrieval and refinement workflows of ColBERTer with vector dimension of 8 or 1 (Uni-ColBERTer). *nDCG and MRR at cutoff 10.*

| Workflow | Model | DL’19-20 nDCG | R@1K | DEV MRR R@1K |
|----------|-------|----------------|------|--------------|
| Retrieval Only Ablation | | | | |
| 1 BOW^2 only | ColBERTer (Dim8) | .323 | .780 | .131 | .989 |
| | Uni-ColBERTer | .280 | .758 | .122 | .880 |
| 3 CLS only | ColBERTer (Dim8) | .669 | .795 | .326 | .958 |
| | Uni-ColBERTer | .674 | .789 | .328 | .958 |
| Single Retrieval > Refinement | | | | |
| 5 BOW^2 > CLS | ColBERTer (Dim8) | .730 | .780 | .373 | .958 |
| | Uni-ColBERTer | .724 | .673 | .369 | .880 |
| 7 CLS > BOW^2 | ColBERTer (Dim8) | .733 | .795 | .375 | .958 |
| | Uni-ColBERTer | .727 | .789 | .373 | .958 |
| Hybrid Retrieval & Refinement | | | | |
| 9 Merge (❼+⩉) | ColBERTer (Dim8) | .734 | .873 | .376 | .981 |
| | Uni-ColBERTer | .728 | .865 | .374 | .979 |

RQ2 What is ColBERTer’s best indexing and refinement strategy?

This study uses ColBERTer with exact matching with 8 and 1 dimensions (Uni-ColBERTer) for BOW^2 vectors, as these are more likely to be used in an inverted index. The inverted index lookup is performed by our hashed id, with potential but highly unlikely conflicts. Then we follow the approach of COIL and UniCOIL to compute dot products for all entries of a posting list for all exact matches between the query and the inverted index, followed by a summation per document, and subsequent sorting to receive a ranked list.

Table 3 presents the results of our study grouped by the type of indexing and retrieval. For all indexing schemes, we use the same trained models. We start with an ablation of only one of the two scoring parts in line 1-4. Unsurprisingly, using only one of the scoring parts of ColBERTer lowers effectiveness. What is surprising, though, is the magnitude of the effectiveness drop of the inverted index only workflow ♂ compared to both using only CLS retrieval (workflow ❼) or refining the results with CLS scores (workflow ⩉). Continuing the results, in the single retrieval then refinement section in line 5-8, we see that once we combine both scoring parts, the underlying indexing approach matters very little at the top-10 effectiveness (comparing lines 5 & 7, as well as lines 6 & 8), only the reduced recall of the BOW^2 indexing is carried over. This a great result for the robustness of our system, showing that it can be deployed in a variety of approaches, and practitioners are not locked into a specific retrieval approach. For example if one has made large investments in an inverted index system, they could build on these investments with Uni-ColBERTer.

Finally, we investigate a hybrid indexing workflow ⩉, where both index types generate candidates and all candidates are refined with the complimentary scoring part. We observe that the recall does increase compared to only one index, however, these improvements do not manifest themselves in the top-10 effectiveness. Here, the results are very close to the simpler workflows ♂ & ⩉. Therefore, to keep it simple we continue to use workflow ♂ and would suggest it as the primary way of using ColBERTer, if no previous investments make workflow ⩉ more attractive.

A general observation in the neural IR community is that more capacity in the number of vector dimensions usually leads to better results, albeit with diminishing returns. To see how our enhanced reduction fit into this assumption, we study:

RQ3 How do different configurations of dimensionality and vector count affect the retrieval quality of ColBERTer?

We must test whether ColBERTer’s reductions of the number of vectors improves effectiveness or reduces costs when compared with merely reducing the number of dimensions. In Figure 3 we show the tradeoff between storage requirements and effectiveness of our model configurations and closely related baselines.

First, we observe that the results of the single vector TAS-B and multi-vector staged pipeline of TAS-B + ColBERT (ours) form a corridor in which our ColBERTer results are expected to reside. Conforming with the expectations, all ColBERTer results are between the two in terms of effectiveness.

Figure 3 displays 3 ColBERTer reduction configurations for 32, 16, 8, and 1 (Uni-ColBERTer) token vector dimensions. Within each configuration, we observe that increased capacity improves effectiveness at the cost of larger storage. Between configurations, we see that removing half the vectors is more efficient and at the same time equal or even slightly improved effectiveness. Thus, using our enhanced reductions improves the Pareto frontier, compared to just reducing the dimensionality. In the case of Uni-ColBERTer, there is no way of further reducing the dimensionality, so every removed vector enables previously unattainable efficiency gains. Our most efficient Uni-ColBERTer with all (BOW^2 and CS) reductions enabled reaches parity with the plaintext size it indexes. This includes the dense index which at 128 dimensions roughly takes up 2/3 of the total space.

6.2 Comparing to Related Work

Fast and complex developments in neural IR make it increasingly difficult to contrast retrieval models, as numerous factors influence
effectiveness, including training data sampling, distillation, and generational training, and it is crucial to also compare systems by their efficiency. We believe it is important to show that we do not observe substantial differences in effectiveness compared to other systems of similar efficiency and that small deviations of effectiveness should not strongly impact our overall assessment, even if those small differences come out in our favor. With that in mind, we study:  

RQ4 How does the fully optimized CoiBERTer system compare to other end-to-end retrieval approaches?

Table 4 groups models by our main efficiency focus: the storage requirements, measured as the factor of the plaintext size.

### Low Storage Systems (max. 2x Factor)

| Model | Storage Factor | Query Latency | Interpret. Ranking | TREC-DL'19 nDCG@10 | R@1K | TREC-DL'20 nDCG@10 | R@1K | DEV MRR@10 | R@1K |
|-------|----------------|---------------|--------------------|---------------------|------|---------------------|------|------------|------|
| 1 [36] BM25 (PISA) | 0.7 GB × 0.2 | 8 ms | ✓ | 0.501 | 0.739 | 0.475 | 0.806 | .194 | .868 |
| 2 [56] JPQ | 0.8 GB × 0.3 | 90 ms | ✗ | 0.677 | – | – | – | 0.341 | – |
| 3 [26] UniCoIL-Tok | N/A | N/A | ✓ | – | – | – | – | 315 | – |
| 4 [36] UniCoIL-Tok (+docT/query) | 1.4 GB × 0.5 | 37 ms | ✓ | – | – | – | – | 352 | – |
| 5 [10, 36] SPLADEv2 (PISA) | 4.3 GB × 1.4 | 220 ms | ✗ | 0.729 | – | – | – | 369 | 0.979 |
| 6 [28] DSR-SPLADE + Dense-CLS (Dim 128) | 5 GB × 1.6 | 32 ms | ✗ | 0.709 | – | 0.673 | – | 344 | – |
| 7 Uni-ColBERTer (Dim 1) | 3.3 GB × 1.1 | 55 ms | ✓ | 0.727 | 0.761 | 0.726 | 0.812 | 0.373 | 0.958 |
| 8 Uni-ColBERTer w. EM (Dim 8) | 5.8 GB × 1.9 | 55 ms | ✓ | 0.732 | 0.764 | 0.734 | 0.819 | 0.375 | 0.958 |

### Higher Storage Systems

| Model | Storage Factor | Query Latency | Interpret. Ranking | TREC-DL'19 nDCG@10 | R@1K | TREC-DL'20 nDCG@10 | R@1K | DEV MRR@10 | R@1K |
|-------|----------------|---------------|--------------------|---------------------|------|---------------------|------|------------|------|
| 9 [12] COIL (Dim 128, 8) | 12.5 GB* × 4.1 | 21 ms | ✓ | 0.694 | – | – | – | 0.347 | 0.956 |
| 10 [12] COIL (Dim 768, 32) | 54.7 GB* × 17.9 | 41 ms | ✓ | 0.704 | – | – | – | 0.355 | 0.963 |
| 11 [28] DSR-SPLADE + Dense-CLS (Dim 256) | 11 GB × 3.6 | 34 ms | ✗ | 0.711 | – | 0.678 | – | 0.348 | – |
| 12 [26, 29] TCT-CoILbertv2 + UniCoIL (+dTSq) | 14.4 GB* × 4.7 | 110 ms | ✓ | – | – | – | – | 0.378 | – |
| 13 CoiBERTer (Dim 16) | 9.9 GB × 3.2 | 51 ms | ✓ | 0.726 | 0.782 | 0.719 | 0.829 | 0.383 | 0.961 |
| 14 CoiBERTer (Dim 32) | 18.8 GB × 6.2 | 51 ms | ✓ | 0.727 | 0.781 | 0.733 | 0.825 | 0.387 | 0.961 |

### 6.3 Out-of-Domain Robustness

In this section we evaluate the zero-shot performance of our CoiBERTer architecture, when it is applied on retrieval collections from domains outside the training data to answer:

RQ5 How robust is CoiBERTer when applied out of domain?

Our main aim is to present an analysis grounded in robust evaluation [50, 58] that does not fall for common problematic shortcuts in IR evaluation like influence of effect sizes [11, 52], relying on too shallow pooled collections [2, 31, 53], not accounting for pool bias in old collections [5, 40, 41], and aggregating metrics over different collections which are not comparable [45]. We first describe our evaluation methodology and then discuss our results presented in Figure 4.

**Methodology.** We selected seven datasets from the ir_datasets catalogue [35]: Bio medical (TREC Covid [49, 51], TripClick [39], NFCorpus [4]), Entity centric (DBPedia Entity [14]), informal language (Antique [13], TREC Podcast [21], news cables (TREC Robust 04 [48]). The datasets are not based on web collections, have at least 50 queries, and importantly contain judgements from both relevant and non-relevant categories. Three datasets are also part of the BEIR [46] catalogue. We choose not to use other datasets from BEIR, as they do not contain non-relevant judgements, which makes it impossible to conduct pooling bias corrections.

We follow Sakai [40] to correct our metric measurements for pool bias by observing only measuring effectiveness on judged passages, which means removing all retrieved passages that are not judged and then re-assigning the ranks of the remaining ones. This is in contrast with the default assumption that non-judged passages are not relevant, which naturally favors methods that have been part of the pooling process. Additionally, we follow Soboroff [45] to utilize an effect size analysis that is popular in medicine and social sciences. Soboroff [45] proposed to use this effect size as meta analysis tool to be able to compare statistical significance across different retrieval collections. In this work we combine the
We apply our models, trained on MSMARCO, end-to-end in a zero-shot fashion with our default settings for retrieval. We compare ColBERTer with a single token dimension and exact matching prior. Only on TREC Robust 04 is the small improved difference inside the 95% CI, including the summary effect model, suggesting to use the more efficient model. We also compare our model to an effective neural dense retriever TAS-B \[16\], shown to work well out of domain \[46\]. We report the effect of using ColBERTer (Dim32) vs. TAS-B in Figure 4c, which paints a less clear image than in the other two cases. Most collections overlap inside the 95% CI, including the summary effect model, suggesting the models are equally effective. Only the Antique collection is significantly improved by ColBERTer. TREC Covid is a curious case: looking at absolute numbers, one would easily assume a substantial improvement but because it only evaluates 50 queries the confidence interval is very wide. Finally, what does this mean for a deployment decision of ColBERTer vs. TAS-B? We need to consider other aspects, such as transparency. We argue ColBERTer increases transparency over TAS-B as laid out in this paper and it does not show a single collection with significantly worse results, favoring the selection of ColBERTer.

**Discussion.** Figure 4a illustrates the effect of using Uni-ColBERTer instead of BM25 across collections and the corresponding summary effect. Compared to the retrospective approach of hypothesis testing with p-values, confidence intervals are predictive \[45\]. Considering the TripClick collection, for example, we expect the effect to be between 0.9 and 2.5% of the time, indicating that we can detect the effect size of 1.7 SMD at the given confidence level and underlining the significant effectiveness gains using Uni-ColBERTer over BM25. Only on TREC Robust 04 is the small improved difference inside a 95% confidence interval. Overall, by judging the summary effect in Figure 4a, we expect choosing Uni-ColBERTer over BM25 consistently and significantly improves effectiveness. Similarly, considering Figure 4b, we expect ColBERTer (Dim32) to consistently and significantly outperform BM25. However, comparing the summary effects in Figure 4a and Figure 4b, we expect Uni-ColBERTer and ColBERTer (Dim32) to behave similarly if run against BM25, suggesting to use the more efficient model. We also compare our model to an effective neural dense retriever TAS-B \[16\], shown to work well out of domain \[46\]. We report the effect of using ColBERTer (Dim32) vs. TAS-B in Figure 4c, which paints a less clear image than in the other two cases. Most collections overlap inside the 95% CI, including the summary effect model, suggesting the models are equally effective. Only the Antique collection is significantly improved by ColBERTer. TREC Covid is a curious case: looking at absolute numbers, one would easily assume a substantial improvement but because it only evaluates 50 queries the confidence interval is very wide. Finally, what does this mean for a deployment decision of ColBERTer vs. TAS-B? We need to consider other aspects, such as transparency. We argue ColBERTer increases transparency over TAS-B as laid out in this paper and it does not show a single collection with significantly worse results, favoring the selection of ColBERTer.

**7 CONCLUSION.**

In this paper, we proposed ColBERTer, an efficient and effective retrieval model that improves the storage efficiency, the retrieval complexity, and the interpretability of the ColBERT architecture along the effectiveness Pareto frontier. To this end, ColBERTer learns whole-word representations that exclude contextualized stop-words, yielding 2.5X fewer vectors than ColBERT while supporting user-friendly query–document scoring patterns at the level of whole words. ColBERTer also uses a multi-task, multi-stage training objective—as well as an optional lexical matching component—that together enable it to aggressively reduce the vector dimension to 1. Extensive empirical evaluation shows that ColBERTer is highly effective on MS MARCO and TREC-DL and highly robust out of domain, while demonstrating highly-competitive storage efficiency with prior dense and sparse models.

**Acknowledgements.** This work has received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement No 822670 and from the EU Horizon 2020 ITN/ETN project on Domain Specific Systems for Information Extraction and Retrieval (H2020-EU.1.3.1.; ID: 860721).
Introducing Neural Bag of Whole-Words with CoBERTer

[55] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. 2020. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. arXiv preprint arXiv:2007.00808 (2020).

[56] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Jiafeng Guo, Min Zhang, and Shaoping Ma. 2021. Jointly optimizing query encoder and product quantization to improve retrieval performance. In Proc. of CIKM.

[57] Giulio Zhou and Jacob Devlin. 2021. Multi-Vector Attention Models for Deep Re-ranking. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing. 5452–5456.

[58] Justin Zobel. 1998. How Reliable Are the Results of Large-Scale Information Retrieval Experiments? In Proc. of SIGIR.