5-2024

GoalBERT: Goal-Directed ColBERT for Iterative Retrieval

Ben Giacalone
bsg8294@rit.edu

GoalBERT: Goal-Directed ColBERT for Iterative Retrieval

by

Ben Giacalone

A thesis submitted in partial fulfillment of the
requirements for the degree of
Master of Science
in Computer Science

B. Thomas Golisano College of Computing and
Information Sciences
Rochester Institute of Technology

May 2024

GoalBERT: Goal-Directed ColBERT for Iterative Retrieval

by

Ben Giacalone

Submitted to the
B. Thomas Golisano College of Computing and Information Sciences
Department of Computer Science
in partial fulfillment of the requirements for the
Master of Science Degree
at the Rochester Institute of Technology

Abstract

We dissect and extend ColBERT, a state of the art multi-vector retrieval model. We first perform a number of experiments to identify the role structural tokens (i.e. `[CLS]`, `[SEP]`, `[MASK]`, `[Q]`) play in retrieval. The most consequential findings are that (1) `[MASK]` tokens can be remapped to their closest non-`[MASK]` embedding without a significant degradation in performance, (2) the existence of `[MASK]`, `[Q]`, and `[D]` does not really affect the contextualization of query text tokens, and (3) `[CLS]` and `[SEP]` can act as "summary" embeddings for the model. In another set of experiments, we extend the number of `[MASK]` tokens to far greater than the number it has been trained with, finding performance to crater when removing all `[MASK]`s, shoot up as the number of `[MASK]`s increases to around 9, then plateau afterwards, with no significant reduction in performance. Using this information, we propose GoalBERT, an iterative retrieval model that uses our findings to select and add weight to terms using a reinforcement learning-based strategy. We compare this model to Baleen, another ColBERT-derived iterative retrieval model, to identify how it performs against the full Baleen pipeline and against just its retrieval component. Though our model underperforms over the baseline, we identify this is due to placing too much responsibility on the retriever, and identify promising future directions to take the research.

Acknowledgments

The second chapter was a paper submitted to the European Conference on Information Retrieval (ECIR) 2024. My thanks to my co-authors of the paper, Greg Paiement and Quinn Tucker. All experiment designs and runs, and most text reported in the second chapter were created by myself, but were informed by an earlier paper written with Greg and Quinn.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

ColBERT [14] is a neural retrieval model that, when released, outperformed other popular neural retrieval models, such as ANCE [37]. Unlike most other contemporary models, it represents documents and queries as multiple vectors instead of a single "summary" vector for each. This is done by simply using every embedding produced by a token. Documents are ranked via the MaxSim operator, in which each query and document embedding is scored by cosine similarity, and the highest score for each query embedding is summed together.

Our goal with this thesis is to train a late-interaction model using reinforcement learning to retrieve documents that maximize some reward function. The approach we use relies heavily on how *structural* tokens in ColBERT are used in scoring, we thus devote much time investigating these tokens. These tokens consist of `[CLS]` and `[SEP]`, tokens that wrap the beginning and endings of queries, respectively, `[Q]`, a token that indicates whether the input corresponds to a query or document, and `[MASK]`, a token used to pad queries that is also used in the document scoring process.

In Chapter 2, we primarily look at `[MASK]` and `[Q]` tokens. We remap all `[MASK]` tokens to their most similar non-`[MASK]` token embedding (as measured by cosine similarity), and find that this causes no significant reduction in performance, indicating `[MASK]` tokens primarily perform term weighting. We also switch the first two words of queries starting with "what is" to the end of the sentence and monitor the change in embedding representation for special tokens, and find that `[CLS]` and `[SEP]` hardly change, while `[Q]` and `[MASK]` tokens dramatically change.

In Chapter 3, we perform additional experiments on `[MASK]` tokens, then study `[Q]`, `[CLS]`, and `[SEP]` in more detail. We find that the `[Q]`, `[D]`, and `[MASK]` tokens have little effect on the contextualization of query text tokens, given that their representations barely change when we remove all `[MASK]` tokens or replace `[Q]` with `[D]` or `[PAD]`. We also show that `[CLS]` and `[SEP]` outperform the first query text token when used in isolation as a single-vector dense retrieval model, indicating these tokens may be aggregating context across the query.

In Chapter 4, we focus more on the `[MASK]` tokens. We extend the number of `[MASK]`s up to 128, far greater than it has been trained with, and notice no significant performance degradation, indicating these tokens simply add weight to the same tokens over and over again. We also see that though removing all MASKs does not cause a huge loss in performance, as the number of MASKs increase, performance quickly shoots upwards, up until the expected number of MASKs are provided. We also migrate from ColBERT v1 to ColBERT v2 [24] and confirm the results of Chapter 2.

In Chapter 5, we introduce our model for comparison, Baleen [13]. Baleen is a late interaction model proposed by the authors of ColBERT that retrieves from a document corpus in multiple steps to verify multi-hop claims. We examine each component of Baleen to understand its approach, and define the multi-hop open domain claim verification task that our proposed system is evaluated on.

In Chapter 6, we propose GoalBERT. Based on our finding that `[MASK]` tokens can be remapped to their nearest non-`[MASK]` embedding without any penalty, we treat the model as a probabilistic query generator, where the cosine similarity of `[MASK]` tokens to query text embeddings are used to parameterize a probability distribution indicating the probability of a weight being added to a token. The system then repeatedly performs retrieval to find evidence to satisfy the query. We compare our approach to Baleen, and although we find that the system performs poorly in comparison, we show that this is due to the problem formulation, not the training strategy, which we show works. We finish by highlighting new research directions based on these results.

Chapter 2

ColBERT: Structural Query Tokens and `[MASK]`

In this chapter, we start our examination into structural tokens in ColBERT, focusing mainly on the `[MASK]` token used to pad queries. Building off our initial results here with `[MASK]`, Chapter 3 then focuses on `[CLS]`, `[SEP]`, `[Q]`, and `[D]`.

The ColBERT [14] retrieval model uses BERT [8] to produce token embeddings for collection ("document") and query passages. Typically, candidate documents are retrieved using dense retrieval on embedded tokens [37, 44], and candidates are then re-scored using the sum of maximum cosine similarities between each query token embedding and its most similar document token embedding (referred to as the *MaxSim* operator). This token-based rescoring generally improves retrieval effectiveness, and is more interpretable than dense retrieval models.

Interestingly, not all tokens used in ColBERT's scoring are text tokens or sub-tokens: some are *structural tokens* that mark locations and segments of a token sequence to be embedded. For example, the token `[CLS]` always appears at the input start, and `[SEP]` marks the end of the query or document text tokens. ColBERT employs a single modified BERT model to create contextualized embeddings for *every* document and query token, including structural BERT tokens. A `[Q]` or `[D]` token appears immediately after the `[CLS]` token, to signify whether a passage comes from a query or document, and query sequences shorter than the input size (e.g., 32 tokens) are padded with `[MASK]` tokens at the end of the token sequence [1]. Document sequences shorter than the expected document length (e.g., 180 tokens) are padded with `[PAD]` instead. Unlike `[MASK]` tokens, `[PAD]` embeddings do not participate in the MaxSim operation. The following two examples illustrate the query and document token input encodings:

Query: `[CLS] [Q] cost of pools swim spa [SEP] [MASK] ... [MASK]`

Document: `[CLS] [D] prices ... swim spa . [SEP] [PAD] ... [PAD]`

In previous work, Formal et al. [10] considers why ColBERT's ranking mechanism tends to outperform standard lexical models such as BM25. In their analysis, they focus on text tokens in queries, and find that tokens with high Inverse Document Frequency (IDF) produce more *exact* matches in ColBERT query/document token alignments (e.g. Q: "pool", D: "pool") while low IDF terms produce more *inexact* matches (e.g. Q: "is", D: "and"). Furthermore, embeddings for low IDF tokens tend to shift position more in the embedding space, and their removal perturbs ranking more than removing high IDF tokens.

MacAvaney et al. [17] find that misspellings harm retrieval in ColBERT more than lexical models (e.g. BM25), and that ColBERT increases document scores if non-relevant content is appended to the end of the token sequence. Curiously, they also find that appending *relevant* terms to the input sequence actually decreases rank scores, even after controlling for document length – perhaps this interferes with the distribution of tokens used for embedding.

The original ColBERT paper proposes that `[MASK]` tokens provide a form of query augmentation, providing both term re-weighting and query expansion. In [33] and [32], the authors demonstrate that `[MASK]` tokens generally do not match document terms not found in the query (i.e., do not approximate query expansion), requiring these terms to be added explicitly. Instead, `[MASK]` tokens primarily weight query tokens. Wang et al. [34] also find that for many ColBERT based models, using *only* special tokens for retrieval (`[CLS]`, `[SEP]`, `[Q]`, `[MASK]`) is nearly as effective as using all token embeddings, even outperforming the case where only low IDF query embeddings are used.

[1] `[MASK]` was originally devised for BERT to represent a "hidden" token in the input, and is used during training for masked token prediction tasks.

In the remainder of this chapter, we consider how structural tokens impact scoring in ColBERT, focusing in particular on `[MASK]`. We present experiments that address the following research questions:

RQ1. Do `[MASK]` tokens perform more than just term weighting?

RQ2. How sensitive are `[CLS]`, `[SEP]`, `[Q]`, and `[MASK]` to query token order?

2.1 Methodology and Experimental Designs

Implementation, Datasets, and Metrics. We use a ColBERT v1 checkpoint provided by the University of Glasgow trained on passage ranking triples for 44k batches,[2] and run experiments on a server with 4 Intel Xeon E5-2667 v4 CPUs, NVIDIA GeForce RTX 2080 Ti GPUs, and 512 GB RAM. We make use of two datasets in our experiments:

1. MS MARCO [3]'s passage retrieval dev set (8.8 million documents, 1 million queries, binary relevance judgements). Each query has at most 1 matching document. We use this dataset when we want to observe statistics for queries (e.g. cosine distance from one query embedding to another).

2. A combined dataset of the test queries from the TREC 2019 [7] and 2020 [6] deep passage retrieval task (99 queries, graded relevance judgements). Collection documents are the same as MS MARCO. We use this dataset for experiments focused upon retrieval quality.[3]

For MS MARCO, documents with score 2 are considered relevant; however, we consider relevance at 1, 2, and 3 to identify the effect of binarization at different relevance grades. We use MRR@10 to characterize the position of top results, MAP to characterize performance for complete rankings, and to complement MAP we use nDCG@k measures ($k \in \{10, 1000\}$) to utilize graded relevance labels from the TREC data.

Experiments. *RQ1: Do [MASK] tokens perform more than just term weighting?*

[2]http://www.dcs.gla.ac.uk/ craigm/ecir2021-tutorial/colbert_model_checkpoint.zip

[3]Running the TREC test queries takes only about 15 minutes to complete using a multithreaded Rust program that will be publicly available for the conference.

We compare ColBERT with a variant where each `[MASK]` token embedding is replaced by its most similar query token embedding. This forces ColBERT to explicitly perform term weighting (i.e. increasing the weight of a term by copying its representation): copying the nearest query embedding cannot introduce new terms, or perform "soft weighting" by slightly increasing the weight of multiple query tokens. We compare retrieval metrics across three conditions: (1) no token remapping, (2) remapping all structural query tokens (`[CLS]`, `[SEP]`, `[Q]`, and `[MASK]`), and (3) remapping only `[MASK]` embeddings, allowing `[MASK]` tokens to also copy the representation of `[CLS]`, `[SEP]`, and `[Q]`. We hypothesize forcing `[MASK]` representations to the closest query embedding causes the model to perform worse, since it removes any ability to add new terms to the query. We use the TREC 2019-2020 dataset for this experiment.

RQ2: How sensitive are [CLS], [SEP], [Q], and [MASK] to query token order?

We wish to study the effect of query token *position* on structural token contextualization. Simply shuffling the text tokens could change the meaning of the query, so to control for semantics we transform queries of the form "what is ..." into "... is what" (only moving the first two text tokens to the end). To further prevent accidental changes in semantics in the altered queries, we only use queries that are 3-8 tokens long (12,513 queries in the MS MARCO dev set fit this criteria). As a baseline, we experiment with the same reordering pattern, but without requiring the first two tokens to be "what is". We then measure the cosine distances between the structural embeddings of the original and altered queries, and hypothesize that `[Q]` and `[MASK]` embeddings will change more than `[SEP]` and `[CLS]` under this reordering.

2.2 Results

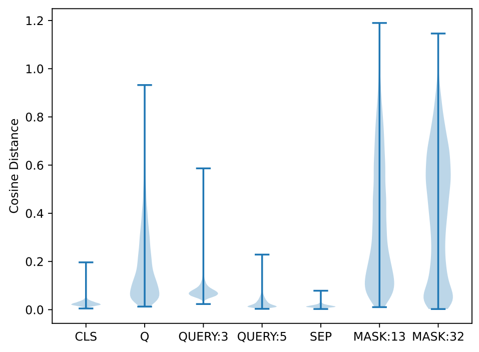
RQ1: Do `[MASK]` tokens perform more than just term weighting? As seen in Table 2.1, replacing *all* structural token embeddings with their closest text token embedding causes a slight reduction in all metrics other than MAP(rel=1), although these are not statistically significant differences. Surprisingly, remapping just `[MASK]` embeddings appears to improve MAP, nDCG@10, and nDCG@1000 slightly over both "None" and "All" conditions. For MAP(rel=1), this is a significant increase (1.5%). This contradicts our hy-

Table 2.1: Replacing structural token embeddings with other query token embeddings (TREC 2019/2020, RQ1). Maximum metric values are in bold; statistically significant differences from "None" are shown with a dagger ($p < 0.05$, Bonferroni-corrected t-tests). "All `[X]`" remaps `[CLS]`, `[SEP]`, `[Q]`, and `[MASK]`.

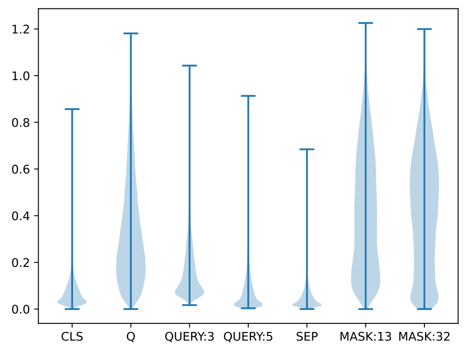
	Structural Token Remapping		
Metric	**None**	**All** `[X]`	`[MASK]`
Binary Relevance			
MAP(rel=1)	0.447	0.454	† **0.462**
MRR(rel=1)@10	**0.930**	0.924	0.923
MAP(rel=2)	0.450	0.444	**0.457**
MRR(rel=2)@10	**0.851**	0.820	0.837
MAP(rel=3)	0.366	0.362	**0.372**
MRR(rel=3)@10	0.557	0.560	**0.563**
Graded Relevance			
nDCG@10	0.689	0.685	**0.694**
nDCG@1000	0.680	0.673	**0.684**

pothesis that remapping `[MASK]` embeddings would harm performance based on previous analyses. It is also interesting because `[MASK]` tokens comprise most of the input sequence when queries are short, meaning that the number of `[MASK]` tokens and the number of query tokens to map to could be impacting the results. For MRR@10, both "All" and "`[MASK]`" conditions are roughly 2-3% lower than the baseline, but this difference is not significant.

RQ2: How sensitive are `[CLS]`, `[SEP]`, `[Q]`, and `[MASK]` to query token order? We compared the shift in `[CLS]`, `[Q]`, `[SEP]`, and two `[MASK]` embeddings before and after we move the first two query tokens to the end and reversing their order. For comparison, we also include the shift in two query text token embeddings as well. In Figure 2.1, QUERY:3 corresponds to the third token in the tokenized query, which in this case is always "what". Right away, we can see distinct differences in how cosine distances are distributed for CLS, SEP, QUERY:3, and QUERY:5 versus Q, MASK:13, and MASK:32. The former group shows barely any shift, while the latter consistently shows large shifts, with higher variation.



(a) "`what is ...`" → "`... is what`"



(b) All queries: "`1 2 ...`" → "`... 2 1`"

Figure 2.1: Distribution of cosine distance $(1-\cos(\vec{e},\vec{e'}))$ for token embeddings before and after query token reordering (MS MARCO, RQ2). For brevity not all tokens are shown, but the general trend of higher variance holds for all `[MASK]` tokens. **Left:** Cosine distances for queries starting with "what is". **Right:** Cosine distances without requiring queries to start with "what is". `[QUERY:3]` and `[QUERY:5]` are the first and third text tokens, respectively; `[MASK:13]` represents the `[MASK]` token at position 13, and `[MASK:32]` represents the final `[MASK]` input token.

As a control comparison, Figure 2.1 also shows what happens when we do not restrict ourselves to queries that start with “what is” — that is, queries like *“airplane flights to florida”* are allowed, which creates the somewhat unnatural ad hoc-style query *“to florida flights airplane”* when shifted, and all tokens show larger representational shifts in this condition; however, we find that `[CLS]` and `[SEP]` and the `[QUERY:3/5]` text token embeddings vary far less than the `[Q]` and `[MASK]` tokens’ representations, which are substantially impacted by the change in query token ordering.

Note that while we tried to avoid semantic change when rearranging tokens in the experiment with “what is” queries, we found that there are still queries that do not have their meanings preserved when performing the switch. For example, a query like *“what is some examples homogeneous”* becomes *“some examples homogeneous is what”*, changing the query from a request for examples of “homogeneous” to asking for the definition of “homogeneous”. When we filtered out queries containing the word “example”, the variance of `[QUERY:3]` dropped from $8.53 \cdot 10^{-4}$ to $7.73 \cdot 10^{-4}$, while the variance of `[Q]` had less of a proportional drop ($2.07 \cdot 10^{-2}$ to $2.06 \cdot 10^{-2}$), indicating some of the variance of non-`[Q]` or `[MASK]` embeddings may be due to these edge cases.

2.3 Discussion

To our surprise, our experiment in which we replaced `[MASK]` tokens by their most similar non-`[MASK]` query token yielded similar effectiveness to standard ColBERT in the TREC 2019/2020 dataset, including a small statistically significant increase in MAP when including weakly-relevant documents in the relevant set. This suggests that we can reduce the number of nearest neighbor lookups for identifying query token matches in documents. For short queries, most of the input to ColBERT is `[MASK]` tokens, and so the number of unique first-stage token retrievals may be a fraction of the full 32-token input size.

A related approach is described by Tonellotto et al. [29], where query embeddings are dropped after contextualization based on frequency statistics. However, rather than pruning a set number of tokens based on collection frequency, we can instead multiply the weight of each query token embedding’s score contribution by the number of `[MASK]` token embeddings most similar to it. This would allow incorporating `[MASK]` embeddings in retrieving candidate documents (as done in this chapter), but without having to consider `[MASK]` tokens when pairing query tokens with document tokens using the *MaxSim* operator.

With respect to interpretability, using `[MASK]` tokens for token weighting simplifies the ColBERT scoring model conceptually as well as computationally. As discussed earlier, the behavior of weighted text terms has been explored extensively in previous work. However, this leaves a question of what role `[MASK]` tokens play in retrieving candidates, which we did not consider in our experiments – all embedded tokens have been used to retrieve candidate documents in our experiments.

To summarize, we have provided new insights into how ColBERT encodes structural tokens for retrieval. Through examination of our first research question, we have found evidence that mask tokens have a very strong relationship with term weighting, although additional study is needed to confirm the exact nature of `[MASK]`'s influence on scoring. Furthermore, in our second experiment we saw that while `[CLS]` and `[SEP]` are relatively insensitive to query text token order, both `[Q]` and `[MASK]`'s representations are highly dependent on the positional encodings assigned to query tokens.

In future work, we would like to examine the role of `[CLS]` and `[SEP]` more closely. As shown in Figure 2.1, the embeddings for these two tokens are far less prone to shift than `[Q]` or `[MASK]`. Initial analysis indicates that `[CLS]` in the query tends to match to `[CLS]` in the document, while `[SEP]` in the query tends to match ending punctuation. In Table 2.1, we also see that allowing `[MASK]` to map to `[CLS]`, `[SEP]`, and `[Q]` generally produces a small improvement in most effectiveness measures, though not to a significant degree for most (with MAP(rel=1) being the exception in our first experiment).

We would also like to gain a better understanding of why mapping `[MASK]` tokens to the nearest query token causes the small improvements in nDCG@10, nDCG@1000, and MAP that we observed. One potential experiment in this direction might be to append different fixed amounts of `[MASK]` tokens to each query. This may reveal if having fewer `[MASK]` tokens causes the model to move them closer to non-`[MASK]` embeddings, or if more `[MASK]` tokens provide more opportunities for term weighting. Understanding this would help produce both more effective and efficient BERT-based retrieval models using late interaction between query and document embeddings.

Chapter 3

Additional Structure Token Experiments

In Chapter 2, we showed how `[MASK]` embeddings primarily perform term weighting, and that `[MASK]` and `[Q]` representations are heavily dependent on the specific order of words in the query. In this chapter, we start by performing some more experiments on `[MASK]` tokens — in particular, how the amount and location of `[MASK]` tokens affect portions of the output. In a similar vein, we then consider how *[D] and [Q]* tokens affect query contextualization. To round out our investigation, we analyze the role of `[CLS]` and `[SEP]`, particularly their ability to aggregate "meaning" across a passage.

We use the following set of new research questions for this chapter to structure our investigation:

- **RQ3: Do `[D]` and `[Q]` tokens modify query representations?**
- **RQ4: Do `[CLS]` and `[SEP]` aggregate information across the whole query?**

3.1 Introducing [PAD]

In the previous section, we describe the structural tokens as `[CLS]`, `[SEP]`, `[MASK]`, `[Q]`, and `[D]`. However, we have neglected to mention one last token: `[PAD]`. This token is effectively a no-op token; it is typically used to pad sequences to a common length during batch inference. The `transformers` library [35] explicitly mentions the attention mechanism masks `[PAD]` tokens such that they do not affect the computed embeddings of other tokens; we also verified that the ColBERT codebase also does this. Thus, `[PAD]` tokens are a useful way of removing a structural token without disrupting the positional encodings of the following tokens (e.g. by moving the tokens afterwards up by one in the sequence).

3.2 RQ3: Do `[D]` and `[Q]` tokens modify query representations?

Given that ColBERT uses a single BERT model for both queries and documents, differing in the input only in whether `[CLS]` is followed by `[Q]` or `[D]`, does ColBERT change how a phrase is encoded when marked as a query or document? As a preliminary analysis, we replace `[Q]` in queries with `[D]`, `[MASK]`, and `[PAD]`. To visualize differences, we project the resulting embeddings onto the first two principal components of the document token embeddings. Afterwards, we check if replacing `[Q]` with `[D]` significantly affects retrieval performance using the TREC 2019/2020 dataset.

To reduce the total number of parameters being trained, ColBERT uses the same model for both query and document encodings. To distinguish between the two use cases, `[unused0]` and `[unused1]` from the BERT WordPiece tokenizer are used as `[Q]` and `[D]` control tokens, respectively.

In Figure 3.1, we show projected query embeddings after `[Q]` is replaced with `[D]` and `[MASK]`. From the figure, we can see that replacing `[Q]` with `[D]` has virtually no effect on how other tokens in the query are contextualized, i.e. how their representations are computed. The projected positions of the query embeddings prior to the switch almost exactly overlap query embeddings after. Even the embedding in position 2, which is usually where `[Q]` goes, does not change its representation after it is replaced by `[D]`. This is in contrast with switching `[Q]` with a `[MASK]` token, also shown in the figure. Though the projected positions are close to how they were embedded prior to the switch, they no longer overlap, as was in the previous case.

Table 3.1 shows the results of running our control setting on TREC versus a setting where D replaces `[Q]` in the query prior to contextualization.

Table 3.1: RQ3: Results for comparing our control setting with a setting where `[Q]` is replaced by `[D]` (TREC 2019/2020 dataset). None of the metric differences are statistically significant (t-test, $p < 0.05$).

	Token Substitution	
Metric	**None**	`[Q]` → `[D]`
MAP	0.447	0.447
nDCG@10	0.689	0.690
nDCG@1000	0.680	0.681
MRR@10	0.930	0.931

There is virtually no change in nDCG@10, nDCG@1000, and MRR@10 when doing this. Thus, it appears that `[Q]` and `[D]` are interchangeable. We also exchanged `[Q]` with `PAD` — effectively removing it from the query — and observed no shift, indicating that similar to the result we got with `[MASK]` tokens in RQ3, these tokens do not effect query text contextualization.

3.3 RQ4: Do [CLS] and [SEP] aggregate information across the whole query?

In BERT's training objective, `[CLS]` is treated as a special token that aggregates whole document context. This is useful for single-vector dense retrievers like ANCE, which map documents and queries to single vector representations that act as "summaries" of the document/query. However, in *ColBERT*'s training process, `[CLS]` is treated as just another query or document token for MaxSim to use. We wanted to understand if in this new role, this token still acts as an information aggregator. We also extend this analysis to `[SEP]`, since it also does not have an explicit meaning in ColBERT's training objective, and it is the structural counterpart to `[CLS]`.

Since ColBERT uses a pretrained BERT model as a base, one hypothesis we have is that `[CLS]` and `[SEP]` act as single-vector dense retrievers. When we examined what `[CLS]` and `[SEP]` tend to match to, we found that much of the time, `[CLS]` in the query matches to `[CLS]` in the document, while `[SEP]` matches to ending punctuation and itself.

To test this, we contextualize TREC queries and test three cases: removing

Table 3.2: Results for comparing our control setting with a setting where `[Q]` is replaced by `[D]`. P-values have been corrected with Bonferroni correction. Statistically significant values have been indicated with a dagger.

SETTING	Just First	Just `[CLS]`	Just `[SEP]`
MAP	0.036	† 0.115	† **0.160**
nDCG@10	0.134	† 0.295	† **0.391**
nDCG@1000	0.115	† 0.258	† **0.329**
MRR@10	0.295	† 0.520	† **0.695**

all query tokens except the first token in the query, removing all query tokens except the `[CLS]` token, and removing all query tokens except the `[SEP]` token. We then compare the performance of each setting to each other. Our hypothesis is that compared to using a single query text token, using just `[CLS]` or just `[SEP]` is significantly better. We use the first query text token since every query is guaranteed to have at least one text token, however we could have also used a random query text token instead.

Table 3.2 shows the results of using only a single query token contextualized by ColBERT. Compared to using the first token, using `[CLS]` or `[SEP]` produces significant improvements across all metrics, indicating that their representations ”summarize” key aspects of the query. Interestingly, though `[CLS]` is explicitly trained to aggregate information over phrases its given, using `[SEP]` seems to be noticeably better. We leave comparing `[CLS]` and `[SEP]` and testing for a significant difference between the two as a future experiment.

While our results indicate using just the `[CLS]` or `[SEP]` embedding as the query performs better than using just the first query text embedding, we were still left wondering why. In particular, we wondered why using just `[SEP]` demonstrated consistent improvements over using just `[CLS]`, especially since `[SEP]` is *not* explicitly trained to aggregate global information at any point.

We thus next test the hypothesis that that `[SEP]` is better than `[CLS]` at intra-passage search, e.g. the `[SEP]` query embedding is better at identifying matches *within* a passage. This idea is based on our observations that `[CLS]` generally matches against `[CLS]` in documents, while `[SEP]` overwhelmingly matches against ending tokens (“.”, “)”, “`[SEP]`”). Since passages can have multiple ending punctuation tokens, while all passages have one `[CLS]` token, in theory, embeddings for these contextualized ending punctuation tokens could be "summarizing" parts of the passage, allowing `[SEP]` to match to these sub-passage summaries.

To test this, we perform the following procedure, visualized in Figure 3.2. For each query in the TREC 2019-2020 dataset, we take the first passage with a relevance score of 2 or higher, take the first passage with a relevance score of 0, then concatenate them together to produce a single conjoined passage. The order in which sub-passage comes first is randomized, and each conjoined passage is truncated to 180 tokens. After contextualization, we remove all characters except periods, to isolate the effect of matching to ending punctuation. Note that passages listed as having 0 relevance often come from the same document as passages with scores of 2-3, making this a particularly challenging task.

As an example, consider the query "where was napoleon born", and the passages "napoleon was born in 1769.", with a relevance score of 3, and "napoleon had 7 siblings", with a relevance score of 0. We concatenate these two passages to form the passage "napoleon was born in 1769. napoleon had 7 siblings.", then run this through the ColBERT document encoder. Afterwards, we separate this concatenated passage back into its constituent sub-passages, obtaining a set of embeddings for each sub-passage. MaxSim is performed between the query embeddings and these embeddings, with the expectation that the relevant passage will be scored higher than the irrelevant passage.

Crucially, conjoined passages share a single `[CLS]` token, but each sub-passage ends with a period. If `[SEP]` truly is aggregating sentence level context by matching to ending punctuation, using just `[SEP]` should have a higher rate of ranking the relevant passage higher than `[CLS]`, since `[CLS]` in the query would match to the conjoined `[CLS]`, while `[SEP]` in the query would match to one of the sub-passage's ending punctuation, making it more fine grained.

We measure performance with all query tokens (baseline), just `[CLS]`, and just `[SEP]`, measuring the accuracy. We also have a setting where we use all document tokens instead of just periods as a baseline.

As shown in Table 3.3, by default, ColBERT is very good at intra-passage retrieval, identifying the correct sub-passage in the joined document 84.5% of the time using all query and document embeddings. When performing retrieval with just periods, however, performance drastically falls across all query

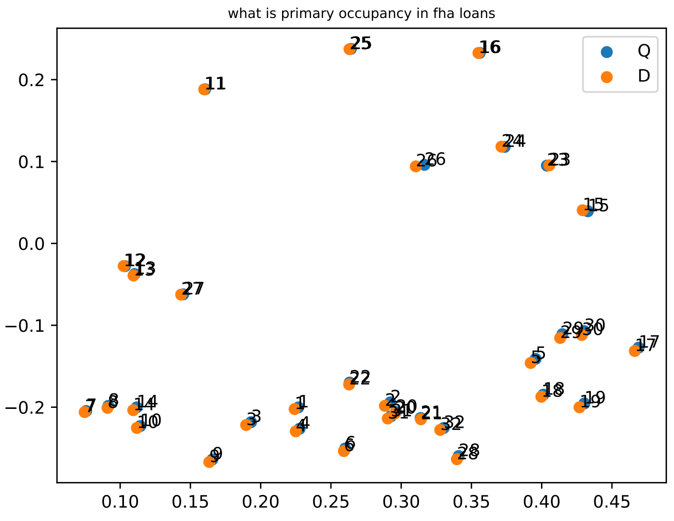
Table 3.3: RQ4: Results from our six settings where we modify the query (keeping all tokens, keeping just `[CLS]`, and keeping just `[SEP]`) and the documents (keeping all tokens and keeping just periods). Each cell corresponds to the percent of time the relevant (rel≥2) passage was selected over the irrelevant (rel=0) passage.

	All Doc. Tokens	Just Periods
All Query Tokens	84.5%	58.8%
Just `[CLS]`	67.0%	57.7%
Just `[SEP]`	50.5%	52.6%

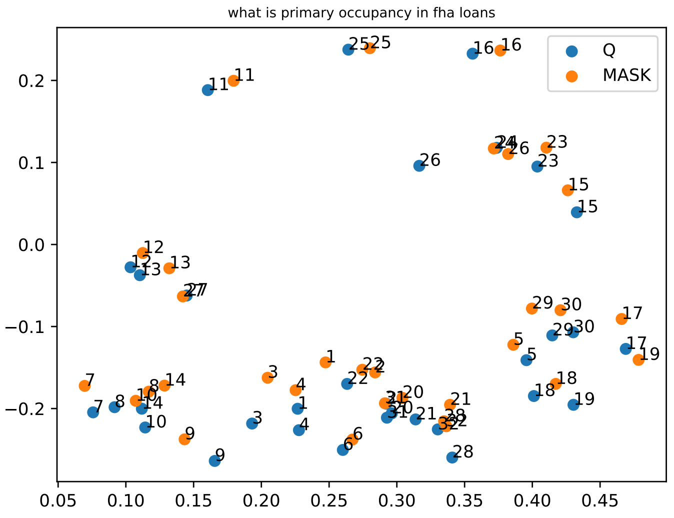
modifications, indicating periods in the document itself do not carry enough representational information about the document to be the reason why `[SEP]` outperforms `[CLS]` in the single-vector dense retriever case. Interestingly, in both document modification settings, `[CLS]` outperforms `[SEP]`, an inversion of what we see in the previous experiment. Based on this evidence, we believe better intra-passage search does not explain the prior experiment's results.

3.4 Summary

In this chapter, we performed additional experiments surrounding structural tokens in ColBERT. Starting with `[Q]` and `[D]` tokens, we analyzed both visually and with metrics what happens if `[Q]` is replaced with `[D]`, finding little effect on the query embeddings, if at all. We also replace `[Q]` with `[PAD]`, effectively removing it, and find that this also has little effect on query embeddings, indicating `[Q]` and `[D]` may not cause the model to treat queries and documents differently. Next, we tried using just `[CLS]` and `[SEP]` when performing retrieval to see if they aggregate information over the entire query, finding that using these two tokens outperforms using the first query text token embedding. However, when we tried to test the theory that this is due to `[SEP]` matching to ending punctuation and `[CLS]` matching to itself, by testing whether `[CLS]` or `[SEP]` is better at intra-passage search, `[CLS]` outperformed `[SEP]`.



(a) Token embeddings in standard ColBERT (Blue) vs. replacing `[Q]` with `[D]` (Orange).



(b) Standard ColBERT (Blue) vs. replacing `[Q]` with `[MASK]` (Orange).

Figure 3.1: RQ4: Illustration of differences in query token embeddings for the query *"what is primary occupancy in the fha loans"* (first two principal components shown). The projected embeddings have been numbered by their position in the query, starting from 0.

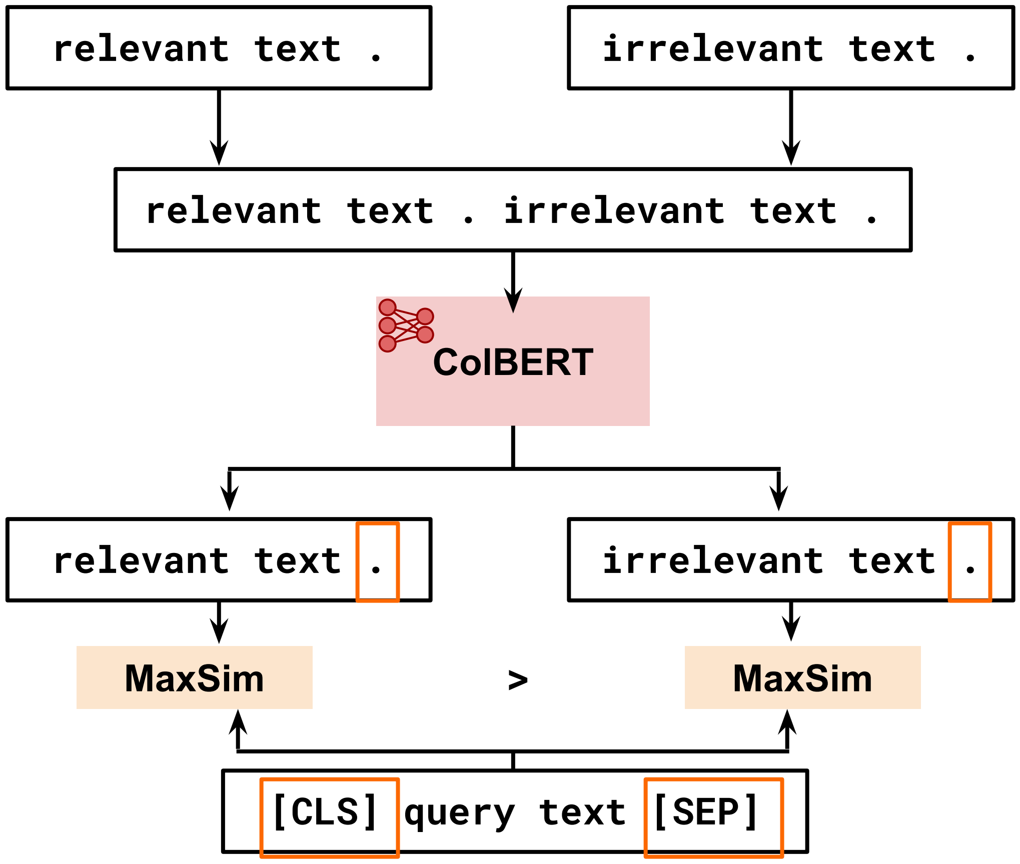


Figure 3.2: A visualization of how contextualization and scoring is performed for RQ4. For a given query, a relevant (rel≥2) and irrelevant (rel=0) passage is concatenated together and contextualized as a single passage. The embeddings for the two passages are then separated again and scored against the query embeddings.

Chapter 4

ColBERT: More Experiments With `[MASK]`

In this chapter, we perform more experiments with the `[MASK]` token. In order to understand how the model's behavior changes as `[MASK]`s are added and removed, we vary the `[MASK]`s from 0 to 128. We also visualize how different `[MASK]`s match to non-`[MASK]` tokens, observing a repeating pattern for weighting certain terms.

ColBERT [14]'s use of multiple token embedding vectors supports fine-grained matching between queries and documents. The model ranks documents by adding the maximum similarity of a document token embedding to each query token embedding, as shown in Equation 4.1. This greedy alignment of query to document token embeddings has been dubbed *MaxSim*.

$$S_{d,q} := \sum_{i \in [|E_q|]} \max_{j \in [|E_d|]} E_{q_i} \cdot E_{d_j}^T \tag{4.1}$$

Here the score for document d given query q, is computed from the set of query and document token embeddings (E_q and E_d, respectively). Embeddings are produced by a BERT-based model [8] finetuned with ColBERT's training objective. For queries, ColBERT prepends a `[Q]` token to indicate a query is being contextualized, and surrounds the tokens with `[CLS]` and `[SEP]` tokens to indicate the beginning and ending of a passage. Finally, the query is padded with `[MASK]` tokens up to a maximum length of 32 tokens. Augmenting the

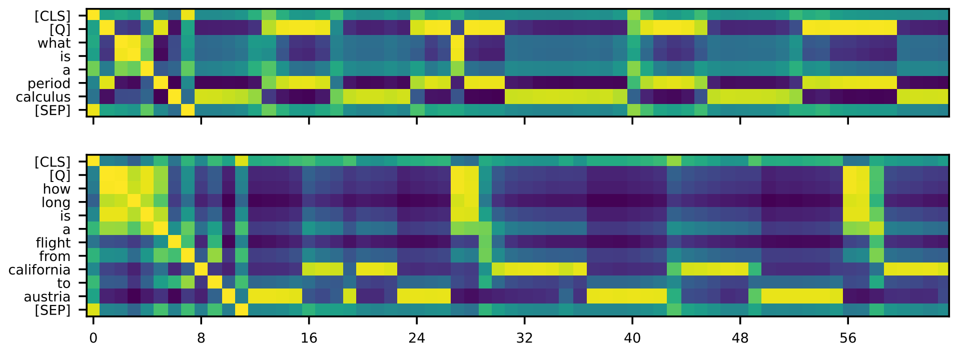


Figure 4.1: Cosine similarity of embedded tokens to each non-`[MASK]` token for positions 0 through 64. A cyclical pattern attending to the most relevant terms in the query (e.g. “period”, “calculus”, “california”, “austria”) can be seen, both before and after 32 tokens (the length trained with).

query with `[MASK]` rather than standard `[PAD]` tokens is key to ColBERT's effectiveness.

In Khattab and Zaharia's original ColBERT paper [14], they show using augmentation with `[MASK]` tokens increases MRR@10 on MS MARCO [3]. Their rationale is that `[MASK]` tokens help introduce new terms to the query, and reweight other query terms. However, later work suggests that `[MASK]` tokens primarily weight other tokens in the query, as summarized in Section 4.1. In this paper we present new experiments to obtain additional insight into how query augmentation maps `[MASK]`s into the contextualized token embedding space. We consider two main research questions:

RQ1. Do `[MASK]` tokens primarily weight non-`[MASK]` tokens in a query when using ColBERTv2?

RQ2. Does effectiveness increase with the number of `[MASK]`s, up to four times the number ColBERT has been trained with?

4.1 Related Work

Prior work has analyzed how ColBERT contextualizes tokens. Formal et. al [10] focused their analysis on query text tokens, using both a model trained with `[MASK]`s and a model finetuned without `[MASK]`s during ranking. They found that query text tokens implicitly capture term importance, because terms with higher IDF tend to produce more exact matches, and change their embedded representation less. When using a model that was finetuned to not use `[MASK]`s, this effect was even more apparent.

Wang et. al [33] considered whether `[MASK]`s in ColBERT actually add new terms to the query, as Khattab et. al [14] proposed in their original paper. They found that it did not, and presented an IDF-based approach for adding new terms to the query. In the same paper, the authors show that `[MASK]` tokens tend to cluster around items already present the query, rather than produce novel query terms, necessitating an approach such as pseudorelevance feedback to add additional query terms.

In Chapter 2, we remapped contextualized `[MASK]` embeddings to their nearest non-`[MASK]` embedding (i.e. `[CLS]`, `[SEP]`, `[Q]`, and the query text tokens), and found no significant difference in MRR@10, nDCG@10/@1000. However, a significant *increase* in MAP was observed both when remapping `[MASK]` vectors to their nearest query text token vector, and when remapping `[MASK]` vectors to their nearest non-`[MASK]` token vector. While interesting, a shortcoming is that our experiments consider only ColBERTv1, instead of the more effective ColBERTv2 [24].

ColBERTv2 uses a more powerful cross-encoding ranker to generate positives and negatives to train with, while ColBERTv1 uses labelled positives and random negatives. This results in an almost 4% gain in MRR@10 on the MS MARCO dev set, allowing it to compete with newer dense retriever models that take advantage of distillation (e.g. PAIR [22]). This may change the behavior of how `[MASK]`s interact with non-`[MASK]` tokens. In our first experiment, we attempt to replicate our earlier results using ColBERTv2.

Tonellotto et. al [29] demonstrated that the number of query token embeddings required for *initial retrieval* in ColBERT can be reduced to as little as 3 by pruning terms frequently present in the collection. They found that `[MASK]`s tend to add less documents to the initial set of documents retrieved, since `[MASK]`s tend to be very similar to existing terms in the documents.

Similar to this paper, in our second set of experiments we perturb the model by modifying the number of `[MASK]` tokens available.

4.2 Methodology

We run our experiments using PyTerrier [18], which contains advanced bindings for ColBERT. Into this framework we load the ColBERT v2 [24] checkpoint provided by the ColBERT team.[1] We confirmed that this checkpoint was trained using the default query length of 32, and that `[MASK]`s had their attention scores zeroed out during training (i.e. no token can attend to a `[MASK]` token during self attention). PyTerrier officially supports only ColBERTv1, but we have verified that the keys PyTerrier expects are also present in our v2 checkpoint.

We do not use v2's index compression, but we believe this is acceptable, since this is not a core feature of the retrieval model. Using the uncompressed index does slightly change performance on MS MARCO from the official metrics. On the MS MARCO dev set, we obtained an MRR@10 of 39.8, Recall@50 of 86.0, and Recall@1000 of 96.2, compared to the official reported metrics of MRR@10 of 39.7, R@50 of 86.8, and R@1000 of 98.4. We suspect this increase in Recall is due to some terms becoming more similar when index compression is applied.

We run our experiments on a server with 4 Intel Xeon E5-2667v4 CPUs, 4 NVIDIA RTX2080-Ti GPUs, and 512 GB RAM. We use our two datasets from Chapter 2:

1. MS MARCO [3]'s passage retrieval dev set (8.8 million documents, 1 million queries, binary relevance judgements). Each query has at most 1 matching document.

2. A dataset combining queries from the TREC 2019 [7] and 2020 [6] deep passage retrieval task (99 queries, graded relevance judgements). Collection is the same as MS MARCO.

As with our experiments in Chapter 2, we use MS MARCO when relevance grades are unimportant, and use the latter when it is, and consider different

[1] `https://downloads.cs.stanford.edu/nlp/data/colbert/colbertv2/colbertv2.0.tar.gz`

relevance levels during evaluation. Additionally, for RQ2, we also use the TREC COVID dataset [30] in addition to the TREC 2019-2020 dataset. This dataset contains 50 queries with graded relevance judgements from 0 to 3. Note that we use the CORD-19 variant [31] instead of the BEIR variant [28] used in the ColBERTv2 paper; thus our baseline measurement differs from the officially reported figure.

RQ1: Do [MASK] tokens primarily weight non-[MASK] tokens in a query when using ColBERTv2? We reproduce our earlier experiments on ColBERT v2, using the TREC 2019-2020 collection. In the first experiment, we compare a baseline of the standard retrieval pipeline against three conditions where certain token embeddings are replaced with others: 1. We remap *all* structural token embeddings (i.e. `[CLS]`, `[SEP]`, `[Q]`, `[MASK]`) to their nearest query text token embedding. 2. We remap `[MASK]` tokens to their nearest non-MASK token (i.e. `[CLS]`, `[SEP]`, `[Q]`, query text tokens). 3. We remap `[MASK]` tokens to their nearest query text embedding, but leave other structural token embeddings (i.e. `[CLS]`, `[SEP]`, `[Q]`) alone.

In the second experiment, we modify all queries in the TREC 2019-2020 collection with a length of 3-8 tokens that start with "what is" by moving these two tokens to the end of the query and swapping their positions (e.g. "*what is* love" becomes "love *is what*"). As indicated in the original paper, this avoids changing query semantics, while shifting the position of every query token. We check the change in cosine distance for `[CLS]`, `[SEP]`, `[Q]`, the first and third query text token, and the 13th and 32nd token in the query, which are guaranteed to be `[MASK]` tokens. As a baseline, we repeat the same experiment without requiring queries to start with "what is", possibly generating nonsense (e.g. "*cost of* swim spa" becomes "swim spa *of cost*").

RQ2: Does effectiveness increase with the number of `[MASK]`s, up to four times the number ColBERT has been trained with? As shown in Figure 4.1, when extending the maximum length of a query past the 32 token window it was trained with, we see a repeating pattern of cosine similarities between `[MASK]` and non-`[MASK]` tokens. It appears that BERT keeps outputting the same weighting pattern for longer query lengths. A natural question then, is how ColBERT fares when the maximum query length is increased, and `[MASK]`-based term weighting dominates document scoring. One may be wary of the unintentional effects of changing `[MASK]` counts this way. For instance, could adding an extra `[MASK]` to the end of a query cause the previous `[MASK]`s, or even the query text tokens, to change their representations in response?

An easily missed detail about ColBERT is that it treats `[MASK]` and non-`[MASK]` tokens differently during the contextualization process — `[MASK]` tokens cannot be attended to during self attention[2]. This has two interesting consequences. One, adding or subtracting `[MASK]`s cannot affect how non-`[MASK]` tokens are contextualized. Non-`[MASK]` tokens cannot attend to `[MASK]` tokens, thus removing `[MASK]`s from the query entirely will not change any of the non-`[MASK]` representations. Two, each `[MASK]` token's computed representation cannot be affected by the existence of *other* `[MASK]` tokens. Each `[MASK]` token can only look at the query and itself, thus, the only change to scoring when adding or removing a `[MASK]` token is the existence of the token's score. In other words, other tokens cannot change their representations in response to to different numbers of `[MASK]` tokens.

In our second experiment, we vary the maximum length of the query from 0 to 96 in steps of two, and measure the resulting performance on TREC 2019-2020. Since we start from a length of 0, we hypothesize that performance will initially increase greatly with each additional `[MASK]`, reflecting the importance of query augmentation. Performance will then plateau, even as more `[MASK]`s are added than seen during training, as the `[MASK]`s repeatedly perform a similar term weighting.

Separately, we report nDCG@10 and nDCG@1000 when the maximum query length is set to 32 to 128, to identify the effect of increasing the total number of tokens seen for each query. In addition to the TREC 2019-2020 dataset, we also use the TREC COVID dataset for this experiment.

ColBERT performs ranking in two phases: an initial set retrieval phase, where documents with at least one embedding very similar to a query embedding are fetched, and a subsequent reranking phase, where documents are reranked by MaxSim. In all experiments, we report metrics for (1) only initial set retrieval is modified, (2) only reranking is modified, and (3) both phases are modified.

[2]To our knowledge, this has not been reported in the ColBERT papers.

Table 4.1: Replacing structural token embeddings by other query token embeddings (TREC 2019-2020, RQ1). Maximum values are in bold; significant differences from "None" are shown with a dagger ($p < 0.05$, Bonferroni-corrected t-tests).

Metric	*ColBERTv2: Token Remapping			
	None	**All** `[X]` → **Text**	`[MASK]` → **Text**	`[MASK]` → **Str. & Text**
Binary Rel.				
MAP(rel≥1)	**0.514**	†0.496	†0.508	0.510
MRR(rel≥1)@10	**0.964**	0.958	0.959	0.960
MAP(rel≥2)	**0.502**	†0.489	0.496	0.498
MRR(rel≥2)@10	0.870	0.871	**0.888**	0.874
MAP(rel≥3)	**0.395**	0.388	0.387	0.391
MRR(rel≥3)@10	**0.616**	0.593	0.598	0.605
Graded Rel.				
nDCG@10	**0.749**	†0.733	0.741	0.745
nDCG@1000	**0.712**	†0.691	† 0.702	†0.703

4.3 Results

RQ1: Do [MASK] tokens primarily weight non-[MASK] tokens in a query when using ColBERTv2 For the `[MASK]` remapping experiment, we see that on ColBERTv2, remapping `[MASK]`s causes a consistent decrease in performance (see Table 4.1). For nDCG@1000, all conditions are significantly worse than the baseline. The "All [X] → Text" condition performs worse than any other condition, many times being significantly worse than the baseline. The "`[MASK]` → Str. & Text" condition performs best of the three conditions. This is both consistent with our earlier ColBERT v1 results, and provides

Table 4.2: Changing the maximum length of queries from 32 to 128 with [MASK] padding. Maximum values are in bold; significant differences from "32" are shown with a dagger ($p < 0.05$, Bonferroni-corrected t-tests).

	TREC 2019-2020		TREC COVID	
Metric	32	128	32	128
Only Set Retrieval				
nDCG@10	0.749	0.749	0.612	**0.616**
nDCG@1000	0.712	**0.717**	0.343	†**0.350**
Only Reranking				
nDCG@10	**0.749**	0.739	0.612	**0.640**
nDCG@1000	**0.712**	0.707	0.343	**0.349**
Set Retrieval and Reranking				
nDCG@10	**0.749**	0.743	0.612	**0.643**
nDCG@1000	0.712	0.712	0.343	**0.355**

more evidence for that [MASK] embeddings simply select all non-[MASK]s as candidates for term weighting.

For the query shift experiment shown in Figure 4.2, we see the same pattern reported in our earlier experiments: [Q] and [MASK] tokens vary greatly after "what is" is swapped and moved, while [CLS], [SEP], and query text tokens do not change nearly as much. In fact, with ColBERTv2, this difference is even starker. Given that this is a pattern that has now manifested itself across two separately trained checkpoints, with two different training objectives, we suspect that the [Q] token performs a similar function to [MASK] tokens – adding weight to certain tokens to influence scoring.

This would also explain the pattern demonstrated by the [Q] token in Figure 4.1, where [MASK]s that are very similar to the [Q] token are always also very similar to some other token. When we visualized several different queries using the same visualization shown in Figure 4.1, we saw that [Q] was the only non-[MASK] structural token consistently very similar to query text tokens.

RQ2: Does effectiveness increase as the number of [MASK]s increases up to four times the number ColBERT has been trained with?

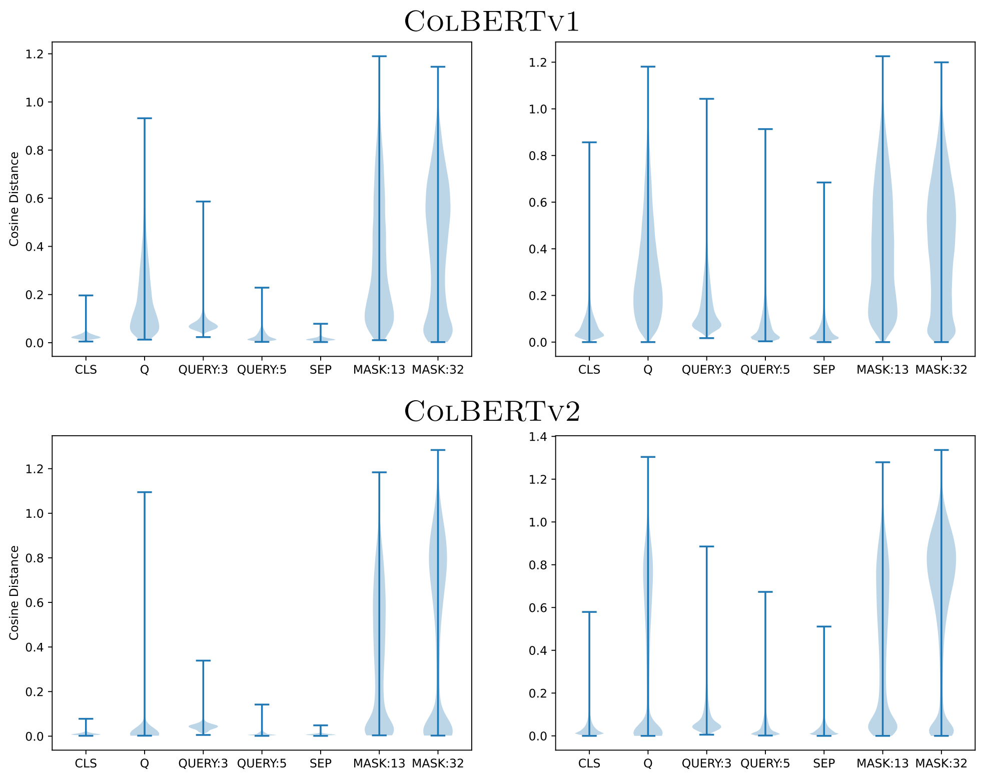


Figure 4.2: Left column: Cosine distance after tokens are switched from ”what is” to ”is what” in ColBERTv1 vs. ColBERTv2. We see the same trend of `[Q]` and `[MASK]` tokens having the most shifting, and an overall increase in shifting when “what is” is not a requirement (right column). The contrast between `[Q]` and `[MASK]` versus other tokens is more apparent in ColBERTv2 than ColBERTv1.

In Table 4.2, we see nDCG@/@1000 on both TREC 2019-2020 and TREC COVID as we vary the maximum query length. We first focus on the results from the TREC 2019-2020 dataset. Modifying only set retrieval causes a minor increase in nDCG@1000, but appears to have no effect on nDCG@10, likely due to baseline set retrieval already retrieving most relevant documents. Modifying only reranking on TREC 2019-2020 causes both nDCG@10/@1000 to decrease. When modifying both phases, nDCG@10 very slightly increases, but nDCG@1000 does not change, likely due to the increase from set retrieval and the decrease from reranking negating each other. Ultimately, all changes observed on TREC 2019-2020 are small, and we never saw an increase or decrease greater than 1%, nor did we observe any statistically significant p-values when performing Bonferroni-corrected t-tests.

On the TREC COVID dataset, we see an increase in nDCG@/@1000 as we increase the length of the query to 128 tokens, for both reranking and set retrieval. These changes are still very small, in the range of 1-3%. The increase in nDCG@1000, however, is statistically significant.

A possible reason for this difference in behavior between TREC 2019-2020 and COVID is that the former dataset has less tokens per query on average compared to the latter (9.68 versus 13.92 tokens), potentially causing certain queries to be incompletely weighted when using only 32 tokens.

In Figure 4.3, we see nDCG@10/@1000, MRR(rel$\geq$2)@10, and MAP(rel$\geq$2) as we vary the number of `[MASK]` tokens each query has. For most of the metrics, moving from 0 to 4 `[MASK]`s appears to actually have a detrimental affect, indicating only using a couple `[MASK]` tokens is worse than none at all. From 4 to $\sim$24 `[MASK]`s, however, we see a sharp increase in nDCG@10/@1000 and MAP(rel$\geq$2). This peak coincides with the point where on average, queries have an overall length of 32 (i.e., the input size used for training). From there on, there is a slight decrease across all metrics, which we expect from the results of the previous experiment. However, we also can see that despite this slight reduction in performance, it is still far better than not having any `[MASK]`s at all.

It appears that as more `[MASK]`s are used on this collection, performance tends to converge to slightly below the baseline. As seen in Figure 4.3, using 8 `[MASK]`s or less causes a statistically significant reduction in performance, while using more than that results in performance that is not significantly different from the baseline. Also, while increasing the number of `[MASK]` tokens

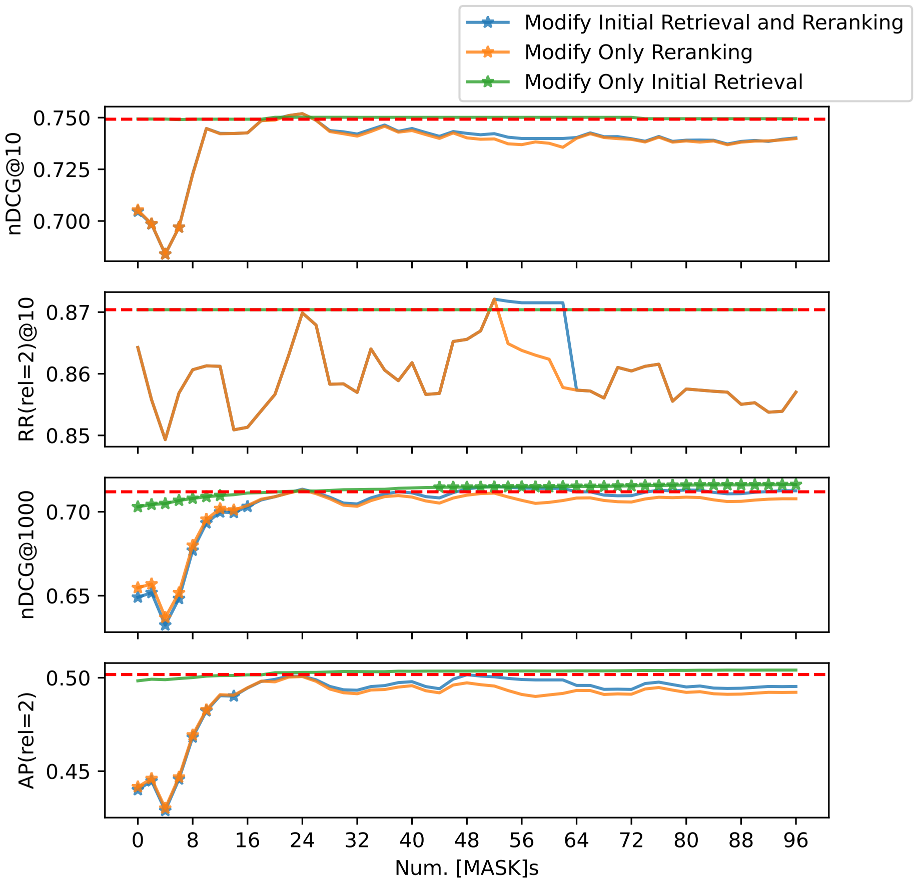


Figure 4.3: nDCG@10, MRR(rel≥2)@10, nDCG@1000, and MAP(rel≥2) increasing number of `[MASK]` tokens from 0 to 96 on TREC 2019-2020. The red line shows a standard length of 32 total tokens. Significant differences from the baseline indicated with a star (Bonferroni correction, $p < 0.05$).

from 0 to 96, RR(rel$\geq$2)@10 does not change in a statistically significant way. For the TREC 2019-2020 dataset, query augmentation does not significantly impact RR(rel$\geq$2)@10.

4.4 Conclusion

The unconventional decision to have ColBERT integrate the padding token used for queries (`[MASK]`) directly into its scoring mechanism has resulted in state of the art performance. Padding with `[MASK]` tokens has been demonstrated to act analogous to term weighting, making it more important for documents to match against some terms than others. An interesting aspect of `[MASK]` representations is that they form a repeating pattern, even when expanding the query past the maximum query length trained with.

We were able to confirm our findings from Chapter 2 on ColBERTv1, showing that even with ColBERTv2, remapping `[MASK]`s to their nearest non-`[MASK]` generally produces non-significant differences in effectiveness metrics, and that `[MASK]`s are much more sensitive to token order than `[CLS]`, `[SEP]`, and even query text tokens. We also found that a partial term weighting using fewer `[MASK]` tokens than used in trained causes effectiveness to decrease, i.e. using no `[MASK]`s performs better than using a small number of `[MASK]`s. Increasing the number of `[MASK]`s from this low point to the amount trained with causes performance to shoot up. Afterwards, performance slightly reduces as `[MASK]`s are added across most metrics, but still performs much better than not using `[MASK]`s at all.

Overall, though there is a slight drop in performance, ColBERT's `[MASK]`-based term weighting strategy performs well past the maximum query length it was trained with, converging to near baseline levels as the size of the query input increases.

In Chapter 6, we build off of the findings of the previous chapters to propose GoalBERT, a ColBERT-derived iterative retrieval system. Our experiments with `[MASK]` embeddings have shown us, first and foremost, that it is valid to view ColBERT as a learned term-weighting model, where the position of `[MASK]` tokens in latent space determine the weight of query text tokens. We have also shown that modifying the number of `[MASK]` tokens does not change query text token embeddings, giving us free reign to add as many as we want, with only a small penalty for any loss of granularity. Finally, ColBERT demonstrating passable results at intra-passage search gives us the ability to chunk documents into short spans after being run through the model, and still be able to recover relevant information.

Chapter 5

Baleen

We now introduce Baleen. As a late interaction-based multi-hop claim verification model, it is the ideal baseline to compare our approach against. In the process, we also define the multi-hop open domain claim verification task, which we use in Chapter 6.

5.1 Task

Baleen targets the family of multi-hop open domain reasoning tasks. Open domain tasks require answering questions from any sort of domain, often relying on a large external corpus to incorporate knowledge from. For tasks such as passage/document retrieval and (single-hop) question answering, retrieval only needs to be performed once. In contrast, *multi-hop* retrieval requires repeated retrieval from a corpus, using items retrieved in previous iterations to guide future hops. For example, in multi-hop question answering, the retriever aims to collect a set of documents that can be used to answer the query, where each document in isolation cannot provide the full answer. While approaches have been developed that use heuristics to non-parametrically identify documents to retrieve [40], we are more interested in approaches that learn to perform this retrieval.

A typical formulation is to have a set of initial documents retrieved using just the query, use the retrieved documents to reformulate the query, then retrieve again, repeating the process across multiple hops [19]. Feldman et al. [9] propose performing query reformulation in latent space using a dense retriever, modifying the dense representation after each hop using retrieved documents. Xiong et al. [38] also use retrieved documents to reformulate a dense query representation, integrating beam search to search for promising passage candidates. Yadav et al. [41] modify the query textually, using heuristics to add unseen terms to the query after each hop.

The specific tasks Baleen addresses are question answering and claim verification. For question answering, the system retrieves a set of documents that allow it to answer a question. For claim verification, the system retrieves a set of documents that provides enough evidence for a classifier to then determine whether or not a given claim is supported by the provided facts. Both being unable to find documents demonstrating the truth of a claim and finding documents that directly contradict a claim fall under this task. The two datasets used for this are HotpotQA [42] and HoVer [11], respectively. The authors primarily focus on the latter, since they find that performance on HotpotQA is easily saturated – the dataset only requires at most 2 hops to answer a question, while HoVer requires 2 to 4 hops instead, increasing the difficulty of the challenge.

Each item in HoVer contains an unordered set of gold (passage, sentence) pairs, which model facts from passages. Exact Match (EM) and F1 are measured between the retrieved and gold set of facts. HoVer uses abstracts taken from Wikipedia in 2017, thus each document has has around 1-4 sentences. An example query from this dataset is "The plant commonly affected by Rhodococcus fascians is a genus of pine trees. Encyclia is as well.", which is a supported claim that requires 3 hops to answer. Note that HoVer provides the number of hops needed to answer a question for each query, removing the need to predict how many hops to perform.

5.2 The Baleen System

We now describe each of Baleen's components, as seen in Figure 5.1.

To begin, the following is a high level overview of how data moves through the system. On each hop, the current query is run through a *retriever* to produce a set of candidate documents Documents are then run through a 2-stage *condenser* to isolate the most relevant facts (e.g., sentences). These facts are added to the end of the query to form the next context, then the system performs the next hop. After a set number of hops, the query and context

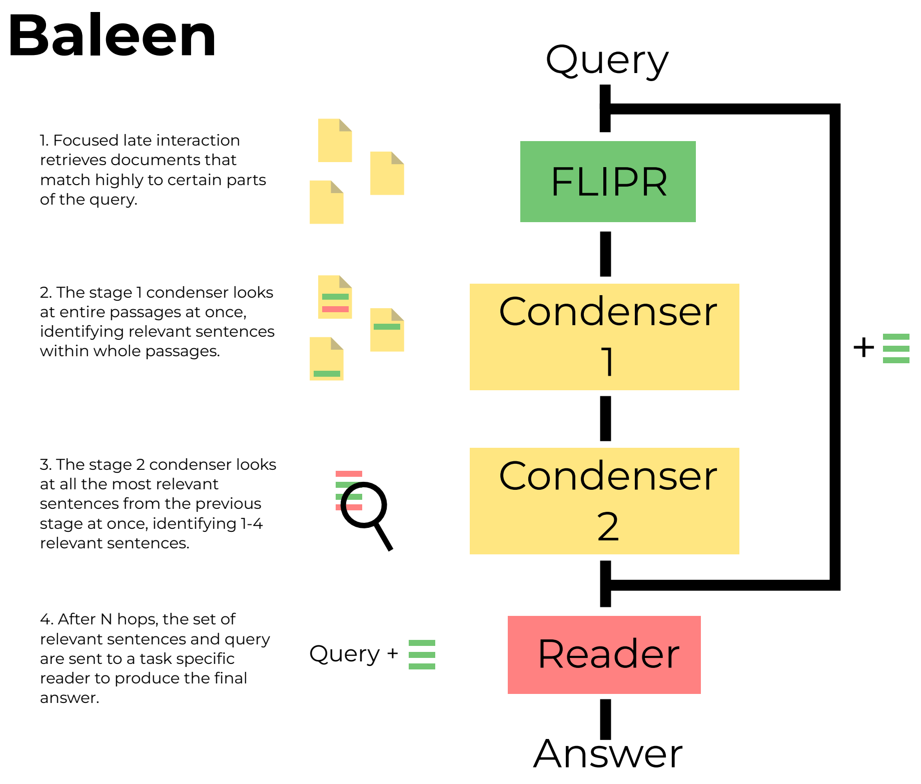

Baleen
Query
1. Focused late interaction retrieves documents that match highly to certain parts of the query.
FLIPR
2. The stage 1 condenser looks at entire passages at once, identifying relevant sentences within whole passages.
Condenser 1
+
3. The stage 2 condenser looks at all the most relevant sentences from the previous stage at once, identifying 1-4 relevant sentences.
Condenser 2
4. After N hops, the set of relevant sentences and query are sent to a task specific reader to produce the final answer.
Query +
Reader
Answer

Figure 5.1: A high level overview of the various components of Baleen. A hop starts with the "Focused Late Interaction Passage Retriever" (FLIPR) using late interaction to retrieve a set of candidate passages. Two rounds of condensing result in a small ($\leq$9) set of sentences that are appended to the query/claim, forming the context for the next hop. The process repeats for 2-4 hops, then the query and context are processed by a reader model that predicts whether the claim is supported or not, based on the retrieved evidence.

are sent to a *reader*, which determines whether or not the query is supported based on the provided context.

For example, in the query "The plant commonly affected by Rhodococcus fascians is a genus of pine trees. Encyclia is as well.", retrieving the entry "Rhodococcus fascians" is required to find plants commonly affected by it. After retrieving the entry for "Rhodococcus fascians" and finding that "Nicotiana" is one such affected plant, the system must retrieve the entry for "Nicotiana" to verify that it is indeed a pine tree. Finally, the entry "Encyclia" is needed to check if it is a pine tree.

All components are implemented with Transformer-based language models. In particular, the retriever uses BERT [8], while the reader and condenser are implemented with ELECTRA models [5]. ELECTRA improves over BERT by learning to discriminate between ground truth and corrupted tokens produced by a generator in the sequences it trains on, resulting in higher performance.

5.2.1 Retriever

As discussed in Chapter 1, the ColBERT retrieval model uses every query and document token for scoring. The score of a document is computed as the sum of the most similar document embedding to each query embedding.

Baleen deviates slightly from this pattern. It proposes *FLIPR*, which stands for *Focused Late Interaction* Passage Retriever. Unlike ColBERT, FLIPR uses only the top-k highest scoring query tokens per document for scoring.

To see why this is helpful, consider the claim "The Statue of Liberty and the Eiffel Tower are located in the same country". To verify this claim, Baleen would have to retrieve articles on the Statue of Liberty and the Eiffel Tower. While there *could* be documents directly linking the two in a single hop (e.g. an article on French Monuments), the HoVer dataset appears to prefer retrieving documents describing separate entities. Because ColBERT uses every query token, using standard ColBERT late interaction would likely retrieve the article for Gustave Eiffel, who contributed to both projects. By only using the highest scoring subset, a single query can match against very different documents, even though they only match highly against certain parts of the query.

In 5.2, we score a query against two different documents, using a k of 3.

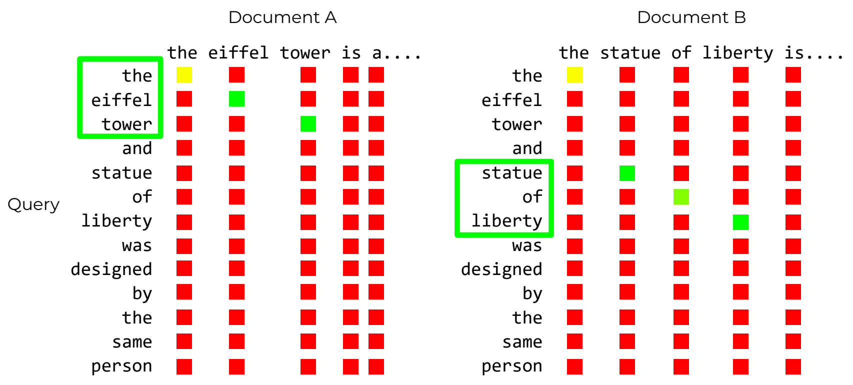
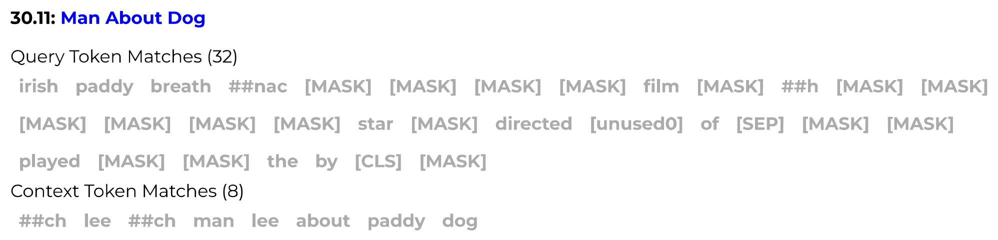


Figure 5.2: An example of FLIPR's scoring mechanism. Each document only matches against the top-k (k=3 here) highest matching tokens in the query.

After finding the maximum cosine similarity for each query token, only the top 3 tokens are used to compute the document's score, indicated with a green box. This allows both the article for the Eiffel Tower and Statue of Liberty to act as highly scoring documents, as opposed to an article that mentions both but does not contain detailed information about each entity (e.g. an article on Gustave Eiffel).

FLIPR computes the score contribution of the query separately from the contribution of the context. By default, 32 tokens from the query are used (half of the 64 tokens that form the query and [MASK]-based padding), while only 8 tokens from the context are used. Figure 5.3 demonstrates which tokens are used for a sample query.

To train FLIPR, Khattab et al. use latent hop ordering. Recall that HoVer provides an *unordered set* of gold passage/sentence pairs. Some of these passages cannot be retrieved based on information in the query alone. In the "George Washington successor" example above, the model would not be able to retrieve the article for "John Adams" before retrieving the article for "George Washington", since it does not know that John Adams succeeded George Washington yet. Since queries are modified after each hop by append-


30.11: Man About Dog

Query Token Matches (32)

irish paddy breath ##nac [MASK] [MASK] [MASK] [MASK] film [MASK] ##h [MASK] [MASK] [MASK] [MASK] [MASK] [MASK] star [MASK] directed [unused0] of [SEP] [MASK] [MASK] played [MASK] [MASK] the by [CLS] [MASK]

Context Token Matches (8)

##ch lee ##ch man lee about paddy dog

Figure 5.3: Query and fact tokens matched for the query "The star of the Irish film directed by Paddy Breathnach played Marcus Agrippa in the HBO drama series 'Rome'." and facts "Paddy Breathnach — He directed 'Man About Dog', 'Blow Dry' and 'Shrooms'." and "Allen Leech — Leech played Marcus Agrippa on the HBO historical drama series 'Rome'.". The document being matched here is for the movie "Man About Dog", which connects the two previously retrieved facts.

ing relevant facts from the previous hop, it is non-trivial to determine what the query on hops after the first hop will look like. Thus, a major challenge is to figure out not just *which* documents to retrieve, but *when* they should be retrieved.

Latent hop ordering extends the idea of weak supervison used in ColBERT-QA [12], proposed by the authors of Baleen. Briefly put, weak supervision is a technique for labeling unlabled data using heuristics. This heuristic can take many forms, from using simple features that exploit biases in the data (such as using anchor text [1]), to using existing models to estimate relevance.

In the single-hop task used in ColBERT-QA, gold passages are not provided, but a short answer string is, allowing for a retrieval heuristic based on BM25. During the first round of training, this BM25 heuristic is used to create positives (highly ranked by BM25) and negatives, then a ColBERT model is trained using these (query, positive, negative) triples. The next two rounds of training then use the ColBERT model trained in the previous round.

This brings us to the multi-hop setting. For the first hop, a ColBERT-QA model is used as a heuristic. Given the query, this retriever will want to retrieve certain gold passages before others. The highest ranked retrieved gold passages are treated as positives, while the non-gold passages are treated as negatives. A set of first-hop queries is produced by adding gold fact sentences present in our first-hop gold passages to queries in the dataset. These first-hop queries can then be used as training data for the next hop.

For the second hop, a standard ColBERT model trained on MS MARCO is finetuned with the positives and negatives for the first hop, and all remaining gold passages for the second hop, to produce a second-hop retriever. The second-hop retriever can now be used to produce positives and negatives with the same procedure used for our first hop. This process continues until a set of positives and negatives for each hop is produced, allowing a model to be trained that works with every hop.

5.2.2 Condenser

After retrieving our initial set of documents, facts (modeled as sentences within retrieved passages, typically 1-4 per passage) are rescored for relevance. In order to scale to as many hops as possible, the model must retain the smallest number of facts that still allows the query to be answered. The condenser consists of two ELECTRA-based models, one which looks at whole passages to score relevant facts within their original context, and another that concatenates all relevant facts together and scores each fact in relation to each other.

The stage 1 condenser looks at entire passages at once. Each sentence in the passage is prepended with a [MASK] token, such that the input is structured as [CLS] [MASK] fact1 [MASK] fact2 ... [SEP]. After passages are contextualized, the special tokens are run through a linear layer to produce relevance scores.

Using the per-hop positives and negatives collected when training the retriever, the model is trained to output high scores for positive sentences, and low scores for negative sentences, using a cross-entropy loss. All sentences that come from a positive passage are considered positives.

After looking at all passages and scoring the sentences within, the top 9 fact sentences are sent to the second stage.

The stage 2 condenser looks at all of the facts (concatenated together) at once. This includes facts currently being used as context. As in the previous stage, each fact is prepended with a [MASK] token and scored. The loss used this time is a linear combination of binary cross entropy loss for each individual fact, and a cross entropy loss for each positive fact against all negatives. This both incentivizes all positive facts to be scored higher than negatives and causes “better” positive facts to be scored higher.

All facts that have a positive score after this step are used as context at the end of each hop. If there are still hops left, this context is added to the query, creating the context for the next hop. Otherwise, the query and context are sent to the reader, where the query and facts are treated as features for a binary classification of whether the query is supported or not.

5.2.3 Reader

The reader is the final stage of Baleen. As with the condensers, it is implemented with an ELECTRA model. Given a query and set of facts, modeled as the query concatenated with all facts (separated by [SEP] tokens), it classifies whether the claim is supported or not. It is trained by associating the outputs of the previous stages with “Supported”/“Unsupported” labels from the dataset. The representation of the [CLS] token is linearly scored to produce a logit for binary classification.

5.2.4 Conclusion

Baleen represents the current state-of-the-art for multi-hop open domain question answering using late interaction By using focused late interaction instead of full ColBERT interaction, documents only need to match against certain parts of the query and context, allowing different kinds of “queries” to be generated within a single hop. Latent hop ordering identifies which documents should be retrieved on which hop, mitigating the need to add hop labels to dataset items. Finally, the 2-stage condenser architecture reduces the multiple passages retrieved into a set of 1-4 sentences to be appended to the query, allowing the system to scale to multiple hops. Due to the way condensing is performed, facts (sentences from retrieved documents) are filtered based on both their context within a passage and their relationship to each other.

While demonstrating strong performance on HoVer, we wonder whether weak supervision is truly the best way to tackle these sorts of tasks. In the next section, we describe a procedure that formulates multi-hop retrieval as a reinforcement learning task, and incorporates our findings on the [MASK] token in ColBERT to train a model to learn to discover which documents are relevant on each step.

Chapter 6

GoalBERT

We now describe a procedure to train late-interaction models using reinforcement learning for iterative retrieval. Our proposed model, GoalBERT, uses the `[MASK]` tokens as actions to select contextualized embeddings in the query and facts, creating a weighted subset of the original query. We compare our final model with Baleen, introduced in Chapter 5.

To review, in addition to tokenizing query and document strings, ColBERT inserts tokens to signal different parts of the input (e.g., `[CLS]` at the beginning and `[SEP]` at the end of query text). The following illustrates how queries and documents are encoded:

Queries: `[CLS] [Q] cost of endless pools swim spa [SEP] [MASK] [MASK]...`
Documents: `[CLS] [D] pool prices range from $ 20 , 000 to $ 30 , 000 [SEP][PAD] [PAD] ...`

While documents are padded to a length of 180 with `[PAD]` tokens, queries are padded to a length of 32 with `[MASK]` tokens. These `[MASK]` tokens are critical to ColBERT's performance. Figure 6.1 shows query embeddings for the query "cost of endless pools swim spa", where `[MASK]` embeddings can be seen congregating three distinct clusters. One cluster contains "pools", "swim", and "spa", capturing the general notion of a pool centered around wellness. Another cluster is centered around "cost", "of", and various structural tokens, capturing the query's request for price related information. Finally, there is a distinct "endless" cluster consisting of only one query text token, capturing some quality of the pool being requested (i.e. being endless).

In Chapter 4, we demonstrated the resilience of the `[MASK]`-based term

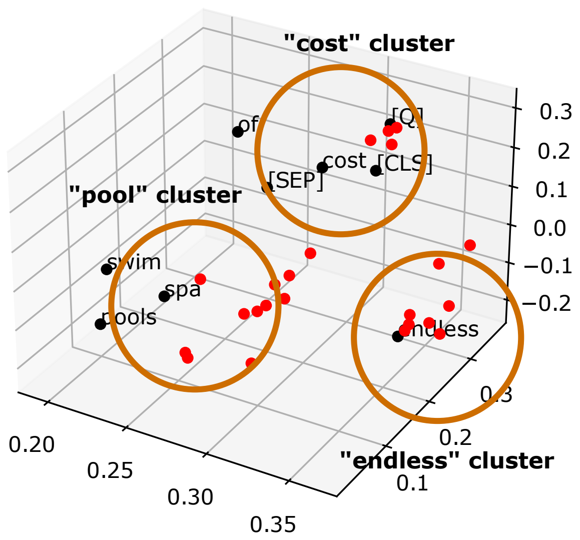


Figure 6.1: PCA projected query embeddings in document space. Circles have been drawn to identify clusters and their associated concepts.

weighting mechanism. We showed that visually, `[MASK]`s tend to cyclically repeat, adding weight to the same non-`[MASK]` tokens over and over again, keeping the ratio of term weights the same as the query increases in length. We confirmed this by extending the maximum length of the query to 128 tokens, where we saw performance plateau but not significantly decrease after the query contained 9 `[MASK]` tokens.

In Chapter 5, we discussed one application of late interaction retrieval outside of document/passage retrieval, Baleen. Given a claim that may or may not be factually supported, Baleen performs multiple hops of retrieval, collecting facts (modeled as sentences) that collectively prove or disprove the claim. Since each passage it retrieves may only be relevant for a part of the query, it proposes focused late interaction, which scores passages using the top-k query embeddings with the highest cosine similarity to the document embeddings (as opposed to using all query embeddings, as with ColBERT).

We would now like to use our findings with the structural tokens from previous chapters to identify ways to modify ColBERT to fit the needs of certain tasks, similar to how focused late interaction was proposed to better fit the needs of multi-hop claim verification. If MASK tokens primarily select and add weight to existing features in the query, can we frame the act of retrieval with this model as a reinforcement learning task? Given the current state (i.e. the current query), the model could be trained to output actions in the form of MASK embeddings that represent a weighted subset of the current query. This would be useful for tasks that require multiple steps of retrieval, such as multi-hop question answering.

6.1 Background

6.1.1 Reinforcement Learning

Reinforcement learning (RL) is an approach to solving Markov decision processes (MDPs) [4] through trial and error [26]. On each timestep, the MDP provides the RL algorithm with the current *state*, and the algorithm outputs a probability distribution over all possible actions it can perform in this state, which is sampled to return an *action* (or set of actions). The MDP then returns the reward, whether the last action brought the MDP to its terminal state, and the next state. A string of experience (sequence of states, actions, and rewards) from a start state to a terminal state is called an episode. The goal of the algorithm is to maximize the expected *return* – the sum of all rewards over the course of an episode – by examining *transitions* (tuples of state, action, reward) collected through interaction with the MDP. The function that takes in a state and returns a distribution over all actions is formally called the *policy*.

Many modern RL algorithms that take advantage of deep neural networks. Broadly speaking, RL algorithms are split into two categories: off policy methods that map state-action pairs to expected returns (e.g. DQNs [20]), and on policy methods that map states to a probability distribution over actions, where actions that lead to higher returns are more likely to be selected (e.g. REINFORCE [27]). The latter are referred to as *policy gradient (PG) methods*.

We are particularly interested in the PG family of methods since we are interested in having our model output a distribution. The original policy gradient algorithm proposed by Sutton et al. [27] simply maximizes the log probability of an action under the policy multiplied by the return of the episode. To cope with the high variance of returns, Konda et al. proposed the actor-critic framework [15], which uses a critic to predict the return produced by a given state, rewarding the policy (the actor) by how much better it performs compared to the critic's baseline prediction. Modern approaches using the actor-critic framework implement both the actor and critic as deep neural networks. Later, Schulman et al. [25] proposed the Proximal Policy Optimization (PPO) algorithm, which uses the ratio of old action probabilities to new action probabilities to determine when the distributions have drifted too far during training, reducing the chances of catastrophic failure during training. We use this algorithm in our approach due to its state of the art performance. Both our policy and critic network are implemented as transformer models, with our policy network being GoalBERT itself.

6.1.2 Reinforcement Learning for Information Retrieval

In recent years, there has been interest in casting tasks in information retrieval as reinforcement learning problems. The benefits are two-fold. First, certain tasks, particularly sequential tasks (i.e. retrieval is performed across multiple steps), are a natural fit for RL. For instance, one may wish to sequentially create a list of *diverse* results for a given query [36] or optimize click rate across multiple pages [43]. Second, RL allows for optimizing non-differentiable objective functions. Metrics such as nDCG can be directly optimized [21, 39] for a given task. Recently, Reinforced Retrieval Augmented Machine Learning (RRAML) [2] has been proposed as a way of learning to retrieve passages in a Retrieval Augmented Generation (RAG) [16] setting, using the output of the whole system to train the retriever. A large motivation of the method we describe is being able to train late interaction models using RL.

6.2 Methodology

6.2.1 GoalBERT

We now describe our proposed model, GoalBERT. Similar to Baleen, GoalBERT performs retrieval across multiple hops. On each hop, the current query and context (except on the first hop, where no context is available yet) is used to retrieve facts. The highest scoring fact not seen yet is appended to the back of the current context, forming the context for the next hop.

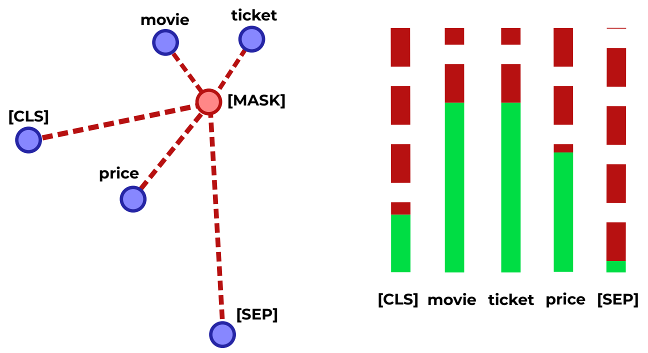


Figure 6.2: The relationship between `[MASK]` embeddings, non-`[MASK]` embeddings, and the probability distributions they generate. The closer a `[MASK]` embedding is to a query text embedding, the higher the probability will be of this embedding being present in the generated query.

GoalBERT uses `[MASK]` tokens to parameterize a distribution over each query and fact embedding, then samples from these distributions once per `[MASK]` to form its actions. Each `[MASK]` is therefore able to select a single non-`[MASK]` embedding. These selected embeddings are then used to query the fact index, a ColBERT v2 index consisting of fact *sentences*.

Figure 6.2 illustrates how term weighting is performed. The query being used here is "movie ticket price", with `[CLS]` and `[SEP]` surrounding the query text, and a single `[MASK]` token being used. After contextualization, the `[MASK]` embedding is very similar to the "movie" and "ticket" embeddings, somewhat similar to `[CLS]` and "price", and very dissimilar to `[SEP]`. The cosine similarities, ranging from 1 to -1, are fed through the softmax function, generating a probability distribution over all non-`[MASK]` tokens, indicated by the green bars.

As shown in Figures 6.2 and 6.3, rather than simply weighting all relevant terms in the original query, we train the model to strategically create sub-queries.

Figure 6.3 demonstrates these steps. Given the statement "Mary Doria Russell and James Joyce are both novelists" (taken from the HoVer [11] claim

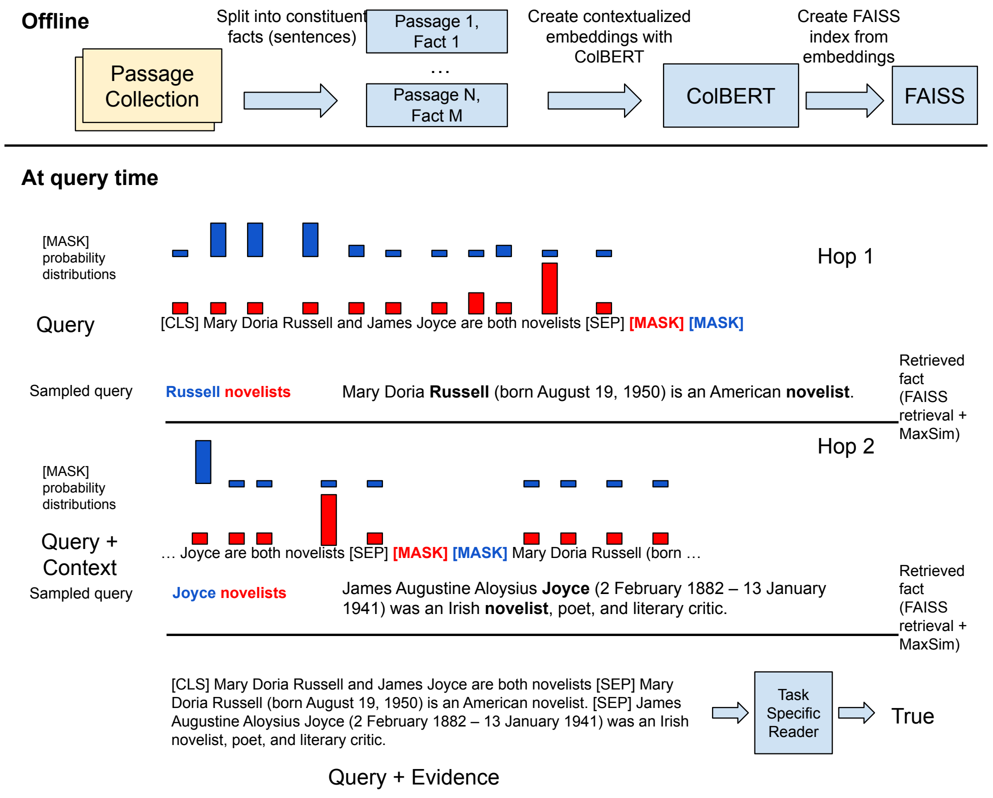

Offline
Passage Collection
Split into constituent facts (sentences)
Passage 1, Fact 1
…
Passage N, Fact M
Create contextualized embeddings with ColBERT
ColBERT
Create FAISS index from embeddings
FAISS
At query time
[MASK] probability distributions
Hop 1
Query
[CLS] Mary Doria Russell and James Joyce are both novelists [SEP] [MASK] [MASK]
Sampled query
Russell novelists
Mary Doria Russell (born August 19, 1950) is an American novelist.
Retrieved fact (FAISS retrieval + MaxSim)
Hop 2
[MASK] probability distributions
Query + Context
… Joyce are both novelists [SEP] [MASK] [MASK] Mary Doria Russell (born …
Sampled query
Joyce novelists
James Augustine Aloysius Joyce (2 February 1882 – 13 January 1941) was an Irish novelist, poet, and literary critic.
Retrieved fact (FAISS retrieval + MaxSim)
[CLS] Mary Doria Russell and James Joyce are both novelists [SEP] Mary Doria Russell (born August 19, 1950) is an American novelist. [SEP] James Augustine Aloysius Joyce (2 February 1882 – 13 January 1941) was an Irish novelist, poet, and literary critic.
Task Specific Reader
True
Query + Evidence

Figure 6.3: The proposed system, GoalBERT applied to fact verification. In this example, the model attempts to verify whether the original query about novelists is true. On each hop, the model contextualizes the current query combined with evidence passages (i.e. the context) and concatenated with some `[MASK]` tokens to produce a set of embeddings. For each `[MASK]` embedding, we generate a probability distribution using the distance from that `[MASK]` embedding to every non-`[MASK]` embedding, then sample these distributions to generate a query. This query is used to retrieve the next piece of evidence. Once the system has performed n number of hops (n is up to the user), the context can be passed to a downstream reader to classify whether the retrieved information verifies a claim or not.

verification dataset), the model is given the task of retrieving relevant documents that would demonstrate the claim to be true or false. On the first hop, the blue `[MASK]` token is contextualized such that it is very close to “Mary”, “Doria”, and “Russell”, while the red `[MASK]` token is very close to “novelists”, thus forming the query “Russell” and “novelists” after randomly sampling these distributions. Identical to ColBERT, these contextualized query embeddings are used to retrieve relevant sentences. The highest scoring sentence is concatenated to the end of the current context, forming the context for the next hop, thus the set of facts grows with each hop. In real-world usage, the query and context is sent to a task specific reader for classifying whether the claim is true or not.

In our experiments, we initialize GoalBERT with the ColBERT v2 weights provided by the ColBERT repository [1]. For our critic network, we use Distilbert [23], a distilled version of BERT. During both training and evaluation, we sample `[MASK]`s, as opposed to selecting the non-`[MASK]` with the highest probability of being chosen for each `[MASK]`.

Action Pruning

As is, the action space of possible queries is still far too large. Consider a query consisting of 30 tokens that uses 20 `[MASK]`s. This result in 1.024e+13 possible action combinations. This inhibits both exploration (the model will not be able to see the effects of most query combinations) and credit assignment (there is high variance due to the extremely large number of final queries that can be generated).

As observed in some figures, some contextualized tokens form clusters in latent space. For example, ”swim”, ”pools”, and ”spa” are all located close to each other in our sample query. Intuitively, if a `[MASK]` token selects any one of the embeddings in this cluster, it will be almost the same as if it had selected the other ones. This leads to a natural way to reduce the scale of the problem: removing ”duplicates” from our action space. We run through the query and identify all embeddings whose cosine similarity is above a certain threshold (we use 0.8) to a previously seen embedding. If so, we mask it out, since selecting this embedding should have a near-identical effect to selecting the previously seen embedding, which we keep. This reduces the set of possible

[1]https://downloads.cs.stanford.edu/nlp/data/colbert/colbertv2/colbertv2.0.tar.gz

actions we can take and increases the probability of the remaining tokens due to normalization. Since retrieved facts by design contain embeddings similar to the query, this results in an almost guaranteed reduction in embeddings to check.

Reward Function

Our reward function closely mirrors the objective of the HoVer dataset. On each hop, our model must retrieve one of the gold facts based on information it currently has access to. Thus, a natural choice is to reward the model whenever it retrieves a gold fact, and give no reward if the fact retrieved is not part of the gold set or has already been seen. Maximizing this objective means returning a relevant fact on every single hop.

However, this function makes for a poor optimization signal. For a four-hop question, if it retrieves anything out of the set of millions of facts except four specific strings, it would get a reward of zero.

Instead, we opt for a softer version of the same idea. For each generated subquery, we retrieve the top 100 facts and identify the rank of the closest unseen fact, using "0" as the rank of the highest ranked item. Our model uses this rank as the reward $(1 - R/100)$, giving us a reward in the range of 0 to 1. This is similar in concept to using the reciprocal rank, except our reward linearly decreases, while the reciprocal rank decreases harmonically. If the closest fact ended up in our top 10 results, we add it to our list of seen facts even though we do not select it for context (we only use the highest ranked fact that hasn't been previously selected when appending to our context), since the model could exploit this by keeping the same gold fact in second place, instead of maximizing the chance of selecting it.

Loss Function

We use the standard PPO clipped loss as our loss function. As typical with PPO, we also incorporate an entropy coefficient to incentivize the network to explore. For our specific problem, we add a term to "freeze" the MASKs and prevent them from changing. Inspired by Distilbert, we add a cosine maximization term that incentivizes the non-`[MASK]` tokens to be as close as possible to the outputs of the original model. The full loss function for the policy at training iteration k, $\mathcal{L}_k$ can be seen in Equation 6.2. The first term

(PPO_k, in Equation 6.1) is the standard clipped PPO loss (using advantage estimation $A^{\pi_k}(s_t, a_t)$), the second term is the cosine maximization loss, and the third term is the entropy loss. N is an operation that computes the non-`[MASK]` embeddings for a state s_t, where each state consists of the query and context at that timestep. N_k corresponds to the non-`[MASK]`s produced by the model at training iteration k, making N_0 the embeddings produced by the original ColBERT model. The actions a_t corresponds to the selected non-`[MASK]`s for each `[MASK]` token. $\pi_k(s_t|a_t)$ represents the probability of selecting actions a_t given state s_t at training iteration k. T is the total number of timesteps. C_m and C_e are coefficients for the `[MASK]` freezing term and entropy term, respectively.

$$PPO_k = \min\left(\frac{\pi_k(a_t|s_t)}{\pi_{k-1}(a_t|s_t)} A^{\pi_k}(s_t, a_t), \quad g(\epsilon, A^{\pi_k}(s_t, a_t))\right) \tag{6.1}$$

$$\mathcal{L}_k = \sum_{t=0}^{T} -PPO_k - C_m \cdot \mathrm{Tr}(\mathrm{N}_0(s_t)^T \mathrm{N}_k(s_t)) - C_e \cdot \pi(a_t|s_t)\log(\pi_k(s_t|a_t)) \tag{6.2}$$

6.2.2 RQ1: How do the number of MASKs impact performance and training stability?

With each MASK token we add, we add another dimension to our action space. This quickly decreases the tractability of the problem, as the effect of changing a single MASK is reduced and the joint probability decreases drastically due to each action probability being independent.

We can alleviate this issue by only using the first n MASKs in the query. Reducing the number of `[MASK]`s reduces the number of possible weighting configurations, potentially hurting the model's ability to perform optimal queries – for example, with only two `[MASK]` tokens, the model can only select a single non-`[MASK]` embedding or two embeddings of equal weight. However, this reduction in `[MASK]`s also dramatically reduces the space of possible embedding selections, making the problem more tractable.

We vary the number of MASKs available from 1 to 4, 6, and all (i.e. all `[MASK]`s are used) and identify the F1 and EM score reached after 490 sampling/training iterations, where during one iteration, we collect a batch of experience with our policy, then train the batch on our model to update action probabilities. We stick to using 2 hops (similar to HotpotQA) instead of the much more challenging 4-hop setting to get faster convergence speeds. We expect performance on the latter setting to be much more difficult when adding more masks, due to the explosion in action space size when more non-`[MASK]` tokens are taken into consideration. Our hypothesis is that similar to our `[MASK]` varying experiments from Chapter 4, performance will sharply increase as `[MASK]`s are increased from 1 to 4. However, due to the greatly increased size of the problem, 6 and all `[MASK]`s will cause the policy to collapse, ultimately resulting in lower performance.

On all runs, we use a policy learning rate of 5e-6, an entropy coefficient of 0.003, a `[MASK]` freezing coefficient of 3, a PPO batch size of 512, and a training batch size of 64.

6.2.3 RQ2: How does GoalBERT compare against Baleen and FLIPR?

We train GoalBERT on the full HoVer train set (i.e. with 2, 3, and 4 hop claims) using 4 `[MASK]` tokens, and compare its performance to Baleen (using the official Baleen checkpoint provided by the GitHub page [2]) on the HoVer dev set. Due to our reliance on using `[MASK]`s, during training, we filter out all queries that leave us with less than 4 `[MASK]`s. We measure the same metrics used in the Baleen paper, sentence and passage level Exact Match (EM) and F1 (combination of precision and recall). EM reports a 1 if our *final* set of retrieved facts or passages is identical to the dataset's gold set of facts or passages, and a 0 otherwise. F1 acts as an intersection-over-union measure between the retrieved and gold set of facts/passages. We report performance across 2, 3, and 4-hop claims individually, along with an average. We hypothesize that due to Baleen's 2 stage condenser architecture, GoalBERT underperforms against the full model.

To compare each model's ability to retrieve highly relevant documents from a wide corpus, we also measure MRR@25 between GoalBERT and FLIPR on the HoVer dev set. GoalBERT is most similar to FLIPR in the Baleen architecture, thus it makes the most sense to compare them head to head.

[2]https://downloads.cs.stanford.edu/nlp/data/colbert/baleen/hover.checkpoints-v1.0.tar.gz

For initial retrieval, ensuring highly relevant documents appear is the priority, while precision takes a backseat. However, if too wide of a net is cast, later parts of the pipeline suffer due to many distractors. We hypothesize that due to GoalBERT being trained to only rank a single document as high as possible on each step, effectively performing a single query per hop, it outperforms FLIPR, whose focused late interaction mechanism causes a query to fulfill many information needs at once (greatly improving recall of gold facts per hop, but potentially reducing precision). We use MRR@25 since Baleen retrieves 25 passages per hop.

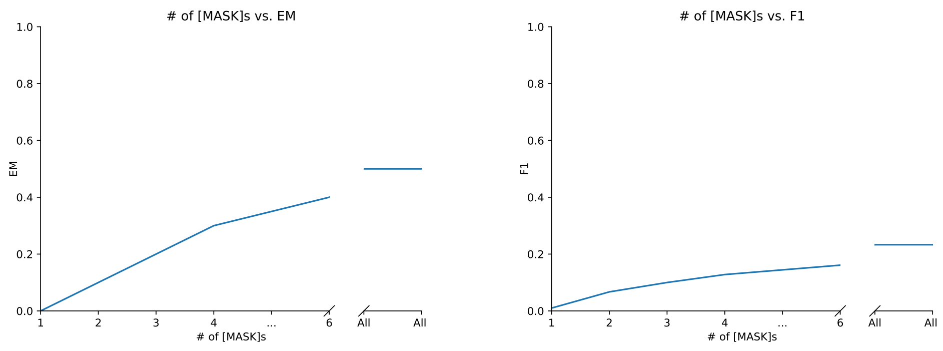
Due to Baleen's usage of condensing after initial retrieval, FLIPR will inherently get better facts as context during retrieval, leading to an unfair comparison. To mitigate this, we use a "perfect" condenser for GoalBERT that takes the results of its retrieval, finds the highest ranked *gold* fact (as opposed to greedy selection of the highest ranked *retrieved* fact), and appends this to the context for the next hop. To account for the fact that FLIPR operates on whole passages, while GoalBERT operates on just sentences, we replace the set of sentences retrieved by GoalBERT with their passages, then remove duplicates. For example, given passages A, B, C, and facts $A_{0..3}, B_{0..3}, C_{0..3}$, where A_0 is the first sentence of A, a ranking of A_0, C_1, A_1, A_2, B_2 would reduce to A, C, B, since a passage from A is returned first, then C, then B.

For each experiment, we also compare GoalBERT against GoalBERT at the start of training (i.e. freshly initialized with ColBERT v2 weights) in order to measure improvement. We use a policy learning rate of 5e-6, an entropy coefficient of 0.3, a `[MASK]` freezing coefficient of 3, a PPO batch size of 512, and a training batch size of 64.

6.3 Results

6.3.1 RQ1: How do the number of MASKs impact performance versus training stability?

In Figure 6.4, we see sentence-level EM and F1 for 1, 2, 3, 4, 6, and all `[MASK]`s. We see that starting from 1 `[MASK]` onwards, adding subsequent `[MASK]`s initially causes a large rise in performance, however performance quickly plateaus (similar to our results from 4). Interestingly, performance never drops even with a large number of `[MASK]`s (i.e. "All"), we see a monotonic increase in



(a) EM (b) F1

Figure 6.4: Sentence-level EM (a) and F1 (b) when training with 1, 2, 3, 4, 6, and all `[MASK]`s. 2-hop claims from the HoVer dev set are used for evaluation.

performance as more `[MASK]`s are added. To speed up convergence, for RQ2, we only use 4 `[MASK]`s, however our results here indicate that using all `[MASK]`s will not harm performance due to catastrophic policy failure during training.

6.3.2 RQ2: How does GoalBERT compare against Baleen and FLIPR?

In Tables 6.1 and 6.2, we see EM and F1 at the sentence and passage level, respectively, for Baleen, GoalBERT, and GoalBERT at the start of training ("Not Finetuned"). As expected, Baleen greatly outperforms GoalBERT across all measures. With each hop, we see lower EM and F1, reflecting the increasing difficulty of the problem.

Table 6.3 shows MRR@25 between FLIPR and our trained and untrained GoalBERT models. Disappointingly, we see that FLIPR greatly outperforms GoalBERT, with an average increase of around 30 points. On average, GoalBERT obtains an MRR@25 of 16.9, indicating relevant passages appear at the 6th position of the ranking on average. In contrast, FLIPR tends to find relevant passages in the 2nd position. These results indicate that GoalBERT performs poorly even when it is just used for initial retrieval.

Despite poor performance in comparison to Baleen and FLIPR, we do see

	Sentence EM				**Sentence F1**			
Model/# Hops	**All**	**2**	**3**	**4**	**All**	**2**	**3**	**4**
Baleen	**39.2**	**47.3**	**37.7**	**33.3**	**81.5**	**81.2**	**82.5**	**80.0**
GoalBERT	0.4	1.3	0.1	0.0	12.3	20.6	10.2	6.9
GoalBERT (Not Finetuned)	0.3	1.0	0.0	0.0	11.5	19.3	9.8	6.1

Table 6.1: Sentence-level EM and F1 for GoalBERT and Baleen, for dev set claims requiring 2, 3, and 4 hops, as designated by the HoVer dataset. “All” corresponds to average performance across all hops.

	Passage EM				**Passage F1**			
Model/# of Hops	**All**	**2**	**3**	**4**	**All**	**2**	**3**	**4**
Baleen	**63.6**	**75.8**	**62.5**	**52.6**	**89.2**	**90.2**	**89.9**	**86.8**
GoalBERT	0.6	1.9	0.1	0.0	15.4	25.2	13.1	9.0
GoalBERT (Not Finetuned)	0.4	1.2	0.0	0.0	14.6	23.1	13.1	8.0

Table 6.2: Passage-level EM and F1 for GoalBERT and Baleen, for dev set claims requiring 2, 3, and 4 hops, as designated by the HoVer dataset. “All” corresponds to average performance across all hops.

an increase in performance compared to the model prior to finetuning. Between the trained and untrained variants, we see an increase in passage/sentence EM, F1, and MRR@25, indicating that our training strategy *does* improve performance on the task, compared to when we freshly initialize the model with ColBERT weights. While this increase was not significant for EM and F1, a two-tailed t-test showed that the final GoalBERT model significantly ($p < 0.05$) improves MRR@25 on the “all” setting.

6.4 Discussion

The results from the experiment for RQ1 show that the benefit of additional `[MASK]`s follows a log scale – we initially get a large increase in EM and F1, but as more `[MASK]`s are added, we get diminishing returns along with a combinatorial explosion of action combinations. Surprisingly, we continue to see

	MRR@25			
Model/# of Hops	**All**	**2**	**3**	**4**
FLIPR	**57.5**	**67.1**	**58.3**	**51.0**
GoalBERT	16.9	30.7	16.0	10.5
GoalBERT (Not Finetuned)	16.0	28.4	15.5	9.9

Table 6.3: MRR@25 for GoalBERT and FLIPR, for claims requiring 2, 3, and 4 hops, as designated by the HoVer dataset. "All" corresponds to average performance across all hops.

performance increase even when using all `[MASK]`s, indicating this combinatorial action space may not pose as big of a problem as originally envisioned, contradicting the second half of our hypothesis. This may be due to the way we restricted how `[MASK]` probability distributions are calculated; since `[MASK]` and non-`[MASK]` embeddings are normalized, and we use cosine similarity between the two as logits for our softmax, the model can never learn to send a `[MASK]` token so far away from other embeddings that it causes policy collapse.

In RQ2, we see that Baleen and FLIPR both outperform GoalBERT. The latter result contradicts our hypothesis that our training strategy improves performance due to being less "scattershot" than FLIPR, resulting in less irrelevant documents sent downstream.

While disappointing, we wondered how much of this was due to our training method versus our problem formulation. Ultimately, our aim was to have the model learn to move `[MASK]`s in embedding space such that reward-maximizing terms would have a higher chance of being selected. If we succeeded on this front, it would indicate that poor performance on the task is due to how we *applied* the model, not necessarily because the reinforcement learning failed.

Figures 6.5 and 6.6 show PCA-fit embeddings before and after 1000 training iterations, for two different claims. In the first, "Greater Swiss Mountain Dog and Harrier are both dog breeds.", the model must learn to retrieve the passage for "Greater Swiss Mountain Dog", then the passage for "Harrier". Thus, we would expect `[MASK]`s to have moved closer to either of the embeddings that make up these two phrases. In (a) of Figure 6.5, we see that the embeddings for "breeds" and "swiss" are strongly preferred by `[MASK]`s. However, after training, `[MASK]`s that select "breeds" become more similar to the embedding for "swiss", indicating that the model has indeed learned

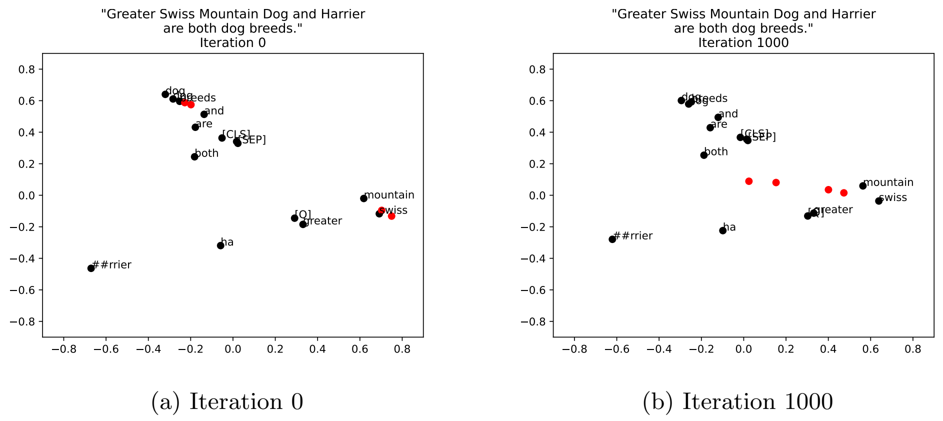


Figure 6.5: Query and [MASK] embeddings before and after 1000 training iterations, for the query "Greater Swiss Mountain Dog and Harrier are both dog breeds.". Embeddings projected with PCA fit on query and [MASK] embeddings from Iteration 0. [MASK] embeddings are in red, while non-[MASK]s are in black.

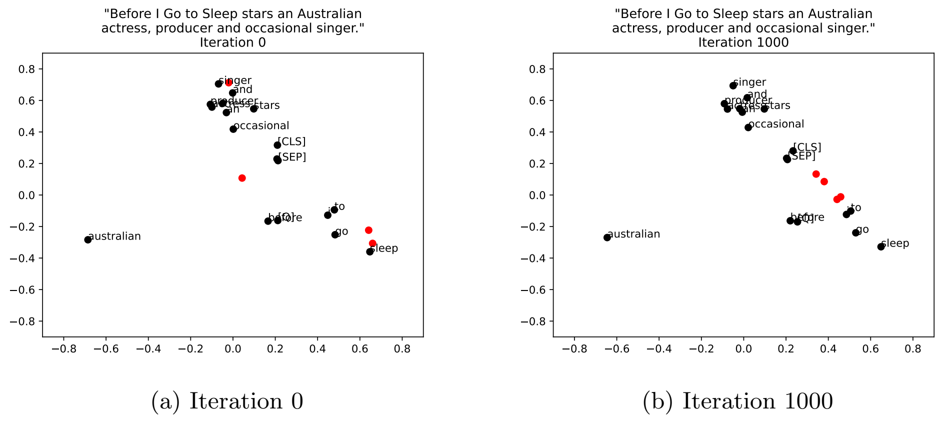


Figure 6.6: Query and [MASK] embeddings before and after 1000 training iterations, for the query "Before I Go to Sleep stars an Australian actress, producer and occasional singer.". Embeddings projected with PCA fit on query and [MASK] embeddings from Iteration 0. [MASK] embeddings are in red, while non-[MASK]s are in black.

that these embeddings tend to match the relevant passage, “Greater Swiss Mountain Dog”.

In Figure 6.6, the claim shown is “Before I Go to Sleep stars an Australian actress, producer and occasional singer.” To answer this query, the model must retrieve the passage for “Before I Go to Sleep”, obtaining information about who starred in the movie. Then, it must retrieve information about the star (Nicole Kidman), to verify that she is an actress, producer, and singer. We would thus expect `[MASK]`s to learn to prefer embeddings that make up the phrase “Before I Go to Sleep”. In (a) of this figure, we see that “singer” is preferred by the `[MASK]`s. However, after training, we do see that `[MASK]`s tend to cluster around the words “before” and “to”, indicating that it has learned that these embeddings tend to retrieve the relevant passage, “Before I Go to Sleep”.

While it does appear that the `[MASK]` embeddings tend to move towards the center of all non-`[MASK]` embeddings, this may be due to our experiment using a particularly high entropy coefficient (incentivizing the model to assign equal probability to all non-`[MASK]` tokens for each `[MASK]`), we do believe that the `[MASK]`s are closer to non-`[MASK]` embeddings that lead to relevant passages than embeddings that do not, as seen in Figure 6.5. Additionally, we have confirmed that with a lower entropy coefficient (e.g. 0.003), the embeddings do move closer to the relevant non-`[MASK]` embeddings.

It is thus our belief that the low EM, F1, and MRR@25 compared to Baleen and FLIPR can be attributed to how we formulated the problem, rather than using RL to train the model modify its `[MASK]` representations to maximize some reward signal. By retrieving facts instead of whole passages, the retriever cannot rely on keywords present across sentences, which could hamper performance. For instance, rather than referring to a subject by name, a sentence may use their pronoun instead. We also greedily select the top fact produced by our ranking directly, instead of using a condensing architecture like Baleen, which would greatly reduce the quality of the context after the first hop. In other words, low performance is due to attempting to perform both initial retrieval and filtering in the same model, which is very difficult.

6.5 Conclusion

Building off of our understanding of ColBERT and its use of MASK tokens, we propose GoalBERT. GoalBERT paramterizes a probability distribution over non-`[MASK]`s for each MASK token, allowing it to probablistically select important terms in the query. This allows us to train it using a reinforcement learning paradigm, where the results of previous queries can impact future rewards. We evaluate this approach using HoVer, a multi-hop claim verification dataset that requires multiple rounds of retrieval to verify claims.

We find that even just using these MASK tokens as actions, the combinatorial space is still far too large to perform effective reinforcement learning. To rectify this, we also propose removing redundant non-`[MASK]` tokens from consideration when parameterizing our action distributions, allowing us to only focus on a select group of important tokens. We also test via RQ1 whether all MASKs are truly necessary for good performance, and find that though performance greatly degrades at 1 MASK token, with each successive token EM and F1 greatly increases, with F1 at 4 MASK tokens obtaining an F1 score 2/3 as high as the full set of MASK tokens, while greatly cutting down on the combinatorial space. At the same time, using all `[MASK]` tokens does not cause catastrophic policy failure, indicating that reducing the number of `[MASK]`s is not strictly necessary.

In RQ2, we evaluate GoalBERT against Baleen and FLIPR. We find that Baleen strongly outperforms our model on every metric (EM, F1) and hop (2, 3, 4, combined), which we expected due to Baleen's condenser architecture providing it with the ability to filter out low quality facts. However, when comparing FLIPR to GoalBERT using MRR@25, we find that even the initial retriever of Baleen greatly outperforms our model. Despite poor performance against the Baleen baseline, we identify an increase across all metrics compared to our freshly initialized model (i.e. using ColBERT v2 weights), indicating our training strategy *is* improving our model.

After further analysis, we find that our model performs the expected behavior of moving `[MASK]` tokens towards relevant query embeddings, indicating that the problem stems from having a single model perform both initial retrieval and fact extraction at once. We thus argue that the increase we observed in our metrics over the course of training demonstrates that using `[MASK]`s to select relevant embeddings in the query *is* a viable strategy for integrating RL into a late interaction framework.

In future work, we would like to follow up on these results by splitting GoalBERT into a passage retrieval model and a sentence filtering model. The retriever would work similarly to GoalBERT as outlined in this chapter, with the exception of retrieving whole passages instead of just fact sentences. For selecting relevant context sentences, we would have a second model that outputs a logit for each passage, similar to the condensers from Baleen. Since the initial retriever and sentence selector can both be trained with RL, we can combine them to create a system that performs end-to-end retrieval and filtering for multi-hop iterative retrieval.

Chapter 7

Conclusion

In this thesis, we have focused on analyzing special tokens in ColBERT and how they contribute to document scoring, in particular, the MASK tokens. We investigated the role of `[CLS]`, `[SEP]`, `[Q]`, and `[MASK]` in how documents are scored, shedding light onto how these tokens contribute to the scoring process. Using our findings, we proposed GoalBERT and evaluated it against Baleen, another late interaction model that performs multi-hop claim verification. Though our final model underperformed compared to Baleen, we argue this was due to our specific problem formulation, and identify new directions of research based on these results.

We started in Chapter 2 by examining the `[MASK]` and `[Q]` tokens. For our first experiment, we remapped all `[MASK]` tokens to their most similar non-`[MASK]` embedding, and found no significant reduction in performance. We also perturbed queries by moving their tokens around in a way that preserved their semantic meaning, and measured the cosine distance between query embeddings before and after perturbation. In doing this, we found that `[CLS]` and `[SEP]` show little change as long as queries retain their semantic meaning, while `[Q]` and `[MASK]` dramatically change, indicating the specific order of words in the query heavily influence the representations of these tokens.

Next, in Chapter 3, we studied `[Q]`, `[CLS]`, and `[SEP]`. The existence of `[Q]` and `[D]` tokens have little effect on how query text tokens are contextualized, given that replacing `[Q]` with `[D]` or `[PAD]` showed no change in query text representations. We also showed that `[CLS]` and `[SEP]` can act as single-vector dense retrievers to a certain degree, as when we performed retrieval with just these tokens in isolation, they outperformed using just the first query text embedding. This built off our results from Chapter 2, in that we again demonstrated how `[CLS]` and `[SEP]` aggregate context across the entire query in ColBERT.

Chapter 4, we returned to examining `[MASK]` tokens. To examine the effect of `[MASK]` tokens on the model, we varied the number of `[MASK]` tokens from 0 to 128 and analyzed its performance. We found no significant performance degradation even at 128, indicating these tokens add weight over and over again to the same tokens in a pattern. We also showed that removing all `[MASK]`s did not cause a huge reduction in performance, although as we added more `[MASK]`s starting from 0, performance greatly increased, plateauing when the number of `[MASK]`s were around the average number of `[MASK]`s seen during training.

In preparation for our final experiment, Chapter 5 introduced Baleen. This late interaction model performs retrieval in multiple steps, reformulating its query after each hop. As we showed, Baleen's usage of focused late interaction and its two-stage condenser model allows it to retrieve a diverse set of relevant documents in initial retrieval, then filter out distracting facts to obtain a small set of relevant facts needed to validate a claim. Along with examining the model and its components, we also defined the multi-hop open domain claim verification task we evaluate our proposed approach on in the next chapter.

In Chapter 6, we finally introduced GoalBERT. Our results from Chapter 2 showed that we could remap `[MASK]` tokens to their nearest non-`[MASK]` neighbor, and our approach was built around this insight. Our proposed system parameterizes a distribution for each `[MASK]` using its cosine similarity to non-`[MASK]`s, then trains the model using reinforcement learning to maximize a reward signal, e.g. the number of gold fact sentences retrieved in multi-hop claim verification. We compared our model to Baleen, and found that it performed poorly in comparison. However, after further examination of the model, we found that the model indeed did maximize our objective function to the best of its ability, moving `[MASK]`s towards relevant query and fact text tokens. Despite these initially disappointing results, we identified new directions of research based on the aspects of this model that succeeded. Overall, we believe that using `[MASK]` distributions to integrate reinforcement learning into late interaction is a promising approach that warrants further research.

Chapter 8

Bibliography

[1] Nima Asadi, Donald Metzler, Tamer Elsayed, and Jimmy Lin. Pseudo test collections for learning web search ranking functions. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, page 1073–1082, New York, NY, USA, 2011. Association for Computing Machinery.

[2] Andrea Bacciu, Florin Cocunasu, Federico Siciliano, Fabrizio Silvestri, Nicola Tonellotto, and Giovanni Trappolini. Rraml: reinforced retrieval augmented machine learning. *arXiv preprint arXiv:2307.12798*, 2023.

[3] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*, 2016.

[4] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[5] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. ELECTRA: pre-training text encoders as discriminators rather than generators. *CoRR*, abs/2003.10555, 2020.

[6] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. Overview of the TREC 2020 deep learning track. *CoRR*, abs/2102.07662, 2021.

[7] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. Overview of the TREC 2019 deep learning track. *CoRR*, abs/2003.07820, 2020.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[9] Yair Feldman and Ran El-Yaniv. Multi-hop paragraph retrieval for open-domain question answering. In Anna Korhonen, David Traum, and Lluís Màrquez, editors, *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2296–2309, Florence, Italy, July 2019. Association for Computational Linguistics.

[10] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. A white box analysis of colbert. In *Advances in Information Retrieval*, pages 257–263, Cham, 2021. Springer International Publishing.

[11] Yichen Jiang, Shikha Bordia, Zheng Zhong, Charles Dognin, Maneesh Singh, and Mohit Bansal. HoVer: A dataset for many-hop fact extraction and claim verification. In *Findings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[12] Omar Khattab, Christopher Potts, and Matei Zaharia. Relevance-guided supervision for OpenQA with ColBERT. *Transactions of the Association for Computational Linguistics*, 9:929–944, 2021.

[13] Omar Khattab, Christopher Potts, and Matei A. Zaharia. Baleen: Robust multi-hop reasoning at scale via condensed retrieval. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 27670–27682, 2021.

[14] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective

passage search via contextualized late interaction over BERT. *CoRR*, abs/2004.12832, 2020.

[15] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[16] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[17] Sean MacAvaney, Sergey Feldman, Nazli Goharian, Doug Downey, and Arman Cohan. ABNIRML: Analyzing the Behavior of Neural IR Models. *Transactions of the Association for Computational Linguistics*, 10:224–239, 03 2022.

[18] Craig Macdonald, Nicola Tonellotto, Sean MacAvaney, and Iadh Ounis. Pyterrier: Declarative experimentation in python from bm25 to dense retrieval. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, CIKM '21, page 4526–4533, New York, NY, USA, 2021. Association for Computing Machinery.

[19] Vaibhav Mavi, Anubhav Jangra, and Adam Jatowt. A survey on multi-hop question answering and generation. *ArXiv*, abs/2204.09140, 2022.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[21] Ali Montazeralghaem, Hamed Zamani, and James Allan. A reinforcement learning framework for relevance feedback. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 59–68, New York, NY, USA, 2020. Association for Computing Machinery.

[22] Ruiyang Ren, Shangwen Lv, Yingqi Qu, Jing Liu, Wayne Xin Zhao, Qiaoqiao She, Hua Wu, Haifeng Wang, and Ji-Rong Wen. PAIR:

leveraging passage-centric similarity relation for improving dense passage retrieval. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Findings of the Association for Computational Linguistics: ACL/IJCNLP 2021, Online Event, August 1-6, 2021*, volume ACL/IJCNLP 2021 of *Findings of ACL*, pages 2173–2183. Association for Computational Linguistics, 2021.

[23] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108, 2019.

[24] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. Colbertv2: Effective and efficient retrieval via lightweight late interaction. *CoRR*, abs/2112.01488, 2021.

[25] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[26] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* A Bradford Book, Cambridge, MA, USA, 2018.

[27] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[28] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021.

[29] Nicola Tonellotto and Craig Macdonald. Query embedding pruning for dense retrieval. *CoRR*, abs/2108.10341, 2021.

[30] Ellen Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R. Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff, and Lucy Lu

Wang. Trec-covid: constructing a pandemic information retrieval test collection. *SIGIR Forum*, 54(1), feb 2021.

[31] Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Doug Burdick, Darrin Eide, Kathryn Funk, Yannis Katsis, Rodney Michael Kinney, Yunyao Li, Ziyang Liu, William Merrill, Paul Mooney, Dewey A. Murdick, Devvret Rishi, Jerry Sheehan, Zhihong Shen, Brandon Stilson, Alex D. Wade, Kuansan Wang, Nancy Xin Ru Wang, Christopher Wilhelm, Boya Xie, Douglas M. Raymond, Daniel S. Weld, Oren Etzioni, and Sebastian Kohlmeier. CORD-19: The COVID-19 open research dataset. In Karin Verspoor, Kevin Bretonnel Cohen, Mark Dredze, Emilio Ferrara, Jonathan May, Robert Munro, Cecile Paris, and Byron Wallace, editors, *Proceedings of the 1st Workshop on NLP for COVID-19 at ACL 2020*, Online, July 2020. Association for Computational Linguistics.

[32] Xiao Wang, Craig Macdonald, and Iadh Ounis. Improving zero-shot retrieval using dense external expansion. *Information Processing Management*, 59(5):103026, 2022.

[33] Xiao Wang, Craig MacDonald, Nicola Tonellotto, and Iadh Ounis. Colbert-prf: Semantic pseudo-relevance feedback for dense passage and document retrieval. *ACM Trans. Web*, 17(1), jan 2023.

[34] Xiao Wang, Craig Macdonald, Nicola Tonellotto, and Iadh Ounis. Reproducibility, replicability, and insights into dense multi-representation retrieval models: From colbert to col*. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 2552–2561, New York, NY, USA, 2023. Association for Computing Machinery.

[35] 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. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing:*

System Demonstrations, pages 38–45, Online, October 2020. Association for Computational Linguistics.

[36] Long Xia, Jun Xu, Yanyan Lan, Jiafeng Guo, Wei Zeng, and Xueqi Cheng. Adapting markov decision process for search result diversification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, page 535–544, New York, NY, USA, 2017. Association for Computing Machinery.

[37] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[38] Wenhan Xiong, Xiang Lorraine Li, Srini Iyer, Jingfei Du, Patrick S. H. Lewis, William Yang Wang, Yashar Mehdad, Scott Yih, Sebastian Riedel, Douwe Kiela, and Barlas Oguz. Answering complex open-domain questions with multi-hop dense retrieval. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[39] Jun Xu, Zeng Wei, Long Xia, Yanyan Lan, Dawei Yin, Xueqi Cheng, and Ji-Rong Wen. Reinforcement learning to rank with pairwise policy gradient. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 509–518, New York, NY, USA, 2020. Association for Computing Machinery.

[40] Vikas Yadav, Steven Bethard, and Mihai Surdeanu. Quick and (not so) dirty: Unsupervised selection of justification sentences for multi-hop question answering. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2578–2589, Hong Kong, China, November 2019. Association for Computational Linguistics.

[41] Vikas Yadav, Steven Bethard, and Mihai Surdeanu. Unsupervised alignment-based iterative evidence retrieval for multi-hop question answering. *arXiv preprint arXiv:2005.01218*, 2020.

[42] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[43] Wei Zeng, Jun Xu, Yanyan Lan, Jiafeng Guo, and Xueqi Cheng. Multi page search with reinforcement learning to rank. In *Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '18, page 175–178, New York, NY, USA, 2018. Association for Computing Machinery.

[44] Jingtao Zhan, Jiaxin Mao, Yiqun Liu, Min Zhang, and Shaoping Ma. Repbert: Contextualized text embeddings for first-stage retrieval. *CoRR*, abs/2006.15498, 2020.