UR^2N: Unified Retriever and ReraNker

Riyaz Ahmed Bhat*, Jaydeep Sen', Rudra Murthy' and Vignesh P*
IBM Research, India
{riyaz.bhat, vignesh.p}*@ibm.com, {jaydesen, rmurthyv}'@in.ibm.com

Abstract

The two-stage retrieval paradigm has gained popularity, where a neural model serves as a re-ranker atop a non-neural first-stage retriever. We argue that this approach, involving two disparate models without interaction, represents a suboptimal choice. To address this, we propose a unified encoder-decoder architecture with a novel training regimen which enables the encoder representation to be used for retrieval and the decoder for re-ranking within a single unified model, facilitating end-to-end retrieval. We incorporate XTR-style retrieval on top of the trained Mono-T5 reranker to specifically concentrate on addressing practical constraints to create a lightweight model. Results on the BIER benchmark demonstrate the effectiveness of our unified architecture, featuring a highly optimized index and parameters. It outperforms ColBERT, XTR, and even serves as a superior re-ranker compared to the Mono-T5 reranker. The performance gains of our proposed system in reranking become increasingly evident as model capacity grows, particularly when compared to rerankers operating over traditional first-stage retrievers like BM25. This is encouraging, as it suggests that we can integrate more advanced retrievers to further enhance final reranking performance. In contrast, BM25's static nature limits its potential for such improvements.

1 Introduction

Retrieval refers to the task of retrieving relevant documents from a larger corpus of documents, given a search string. Retrieval is one of the most active research fields in NLP owing to its many applications such as semantic search (Fazzinga and Lukasiewicz, 2010), Open-domain Question Answering (Voorhees and Tice, 2000), Retrieval Augmented Generation (RAG) (Cai et al., 2019; Lewis et al., 2020; Guu et al., 2020). While there are ongoing research around novel research paradigms such as Splade (Formal et al., 2021), HyDE (Gao et al., 2023), Differential Search Index (Tay et al., 2022), in most of the industry settings, retrieval technologies deployed can broadly be divided into (1) Sparse Retrievers (Robertson and Zaragoza, 2009) (2) Dense Retrievers (Karpukhin et al., 2020; Chang et al., 2019; Guu et al., 2020; Xu et al., 2022; Khattab and Zaharia, 2020; Luan et al., 2021; Santhanam et al., 2022), each paradigm with its own advantages and disadvantages. Sparse retrievers are easy to deploy across domains with fast inference using inverted index, but they heavily rely on keyword based lexical overlap for retrieving relevant documents. On the other hand, dense retrievers also known as neural retrievers are capable of learning embedding vectors which can retrieve text based on their semantic similarity, beyond keyword overlap. However, using dense retrievers come at a higher cost of deployment with dedicated embedding vector stores such as Milvus (Wang et al., 2021), Chroma (Chroma, 2024) etc., higher latency pertaining to more rigorous similarity score computations. Also, dense retrievers often need domain specific tuning and can not generalize to unseen domains as easily as sparse retrievers.

Owing to the complimentary advantages and disadvantages of sparse and dense methods, most industry applications employ a two stage pipeline for practical purposes. Such systems tend to use BM25 like sparse retrievers to do a 1st stage retrieval and then neural models in the 2nd stage to rerank top-k retrieved documents. The neural model used in 2nd stage is also called Reranker (Nogueira et al., 2020; Zhuang et al., 2023a). We posit that having a two-stage approach with a retriever and reranker is a sub-optimal choice, mainly dictated by practical limitations and hardships of developing an end-to-end neural model. Having a separate retriever and a reranker requires maintaining two separate models. Also, because it is a pipeline, none of the components learn from each other and therefore, have a cascading effect in terms of error propagation. If the 1st stage retriever is not retrieving relevant documents in top-k, the reranker can not recover from there. Given these serious limitations, we offer a fresh perspective that both retrieval and reranking should be done by a single unified model.

We propose UR^2N an encoder-decoder based architecture, which is so trained that the encoder representation can be used for retrieval and the decoder from the same model can be used for reranking, thus unifying the retrieval and reranking into a single model. While we want UR^2N to be robust, we have been specifically careful about the practical implications of using UR^2N and performed all our empirical evaluation with 1 GPU only. We build UR^2N on top of mono-T5 reranker (Nogueira et al., 2020) (one could easily replace with another text-to-score reranker such as Rank-T5 (Zhuang et al., 2023a)) and adopt XTR (Lee et al., 2023) style training for the encoder. The choice of XTR is important here as XTR is an optimized version of ColBERT (Khattab and Zaharia, 2020) (discussed in section 4.1) which is a multi-vector based retriever and thus ensures high accuracy for the 1st stage retrieval in UR^2N. We list our contributions categorically as follows:

Our contributions:

- We propose UR^2N, an encoder-decoder architecture that unifies first-stage retrieval and second-stage reranking into a single end-to-end trainable model.
- We build on top of the Mono-T5 reranker (Nogueira et al., 2020) to adopt XTR (Lee et al., 2023) style retrieval to ensure UR^2N is lightweight and easily deployable.
- Empirical results on popular IR benchmark BIER shows that UR^2N performs competitively and even provide gains as a reranker, with the distinct advantage of having a unified single model when compared to the state-of-the-art results by Mono-T5 which is a two-stage process using two different models.
- Empirical results on BIER also shows that the unified modeling in UR^2N also improves the end-to-end retrieval accuracy than the state-of-the-art results by ColBERT and XTR.

We believe that UR^2N opens up a novel paradigm for retrievers where retriever and reranker can be unified into a single model, with its deployment and application easier than current day systems. **We will be releasing the source code and model checkpoints subsequently.**

2 Related Work

We will review the important building blocks which are relevant for our work here. They are (1) Sparse Retrievers, (2) Dense Retrievers and (3) Rerankers.

Among Sparse retrievers BM25 is the most popular and robust sparse retriever which use term frequence (TF) and Inverse Document Frequency (IDF) (Wikipedia, 2024) scores to estimate document relevance given a query. Although it relies on exact keyword matches, due to its simplicity and strong performance across domains and tasks (Thakur et al., 2021), BM25 continues to be a strong baseline for retrievers.

Dense retrievers (a.k.a neural retrievers) use token embeddings obtained from neural language models like BERT (Devlin et al., 2019) etc. and further finetune them for the retrieval task by constrastive finetuning (Izacard et al., 2022). Neural retrievers too can broadly be classified into two types: a) single vector and b) multi-vector approaches. Single vector approaches such as DPR (Karpukhin et al., 2020) obtain a single vector representation for the query and the documents using the [CLS] token representation or mean pooling, which are then used to obtain cosine similarity scores for retrieval. In contrast, multi-vector approaches (Khattab and Zaharia, 2020; Santhanam et al., 2022; Lee et al., 2023) consider each token embedding separately and use a late interaction strategy between query tokens and document tokens for scoring. They apply mathematical heuristics to aggregate over the token level cosine similarity scores to have a final estimate of the query-document similarity. While multi-vector approaches such ColBERT (Khattab and Zaharia, 2020) understandably provide better performance than DPR (Karpukhin et al., 2020), they are much more expensive in terms of storage, compute resource and latency because of token level operations. While ColBERT-v2 (Santhanam et al., 2022) proposed algorithms to reduce the index size, methods like XTR (Lee et al., 2023) proposed novel training and inference methods to greatly reduce the inference latency by considering operations over a bounded subset.

Neural rerankers attempt to overcome the limitation of sparse models of needing exact keyword matches, by using embedding-based neural models to rerank a large number documents retrieved by sparse models. Popular neural rerankers (Nogueira et al., 2020; Zhuang et al., 2023b) are sequence-to-sequence models built over T5 (Raffel et al., 2020) specifically tuned to generate the labels *true* or *false* given a query and document pair. The probability of generating *true* is used as the relevance score to rerank the documents.

3 Motivation for Industry Usage

To motivate the industry need for our work, we take the perspective of a **practising R&D engineer John** who has access to only a limited amount of resources to train, finetune and deploy a robust retriever model. Even though it is already known that with enough training, neural models can do much better than BM25 like sparse models with strict keyword overlap needs, John has to restrict towards a sub-optimal two stage multi model solution where BM25 acts as the core retriever with much cheaper deployment and maintenance cost. A different neural model is used only as a reranker to improve over BM25 search results. This two stage multi-model pipeline not only introduces additional deployment stages but also limits the achievable skyline to the accuracy of the 1st stage retrieval quality.

By designing UR^2N we provide John with a lightweight easy to deploy model which is unified to do both retrieval and reranking using its encoder and decoder. We make design choices that makes UR^2N lightweight for finetuning, with optimized index management. To empirically prove our point adhering to practical resource constraints, we perform all our experiments with only 1 GPU only.

4 UR^2N: System Overview

Our model is based on the XTR model and built on top of Mono-T5. So, before we delve into our architecture, we'll first provide a detailed overview of XTR and Mono-T5 models specifically, the training and inference as an important background. We use the notations and expressions from the original papers while presenting an overview of these.

4.1 UR^2N Background: XTR

XTR is a state-of-the-art multi-vector model which is built over ColBERT, providing huge inference speed up and also improving retrieval performance. For a query $Q = q_{i=1}^{n}$ and a document $D = d_{j=1}^{m}$, where q_i and d_j represent d-dimensional vectors for query tokens and document tokens, XTR uses an alignment matrix between query and document tokens as $\hat{A}_{ij} = \mathbb{1}[j \in \text{topk}_{j'}(P_{ij'})]$, where $P_{ij} = q_i^T d_j$ and the top-k operator is applied over tokens from mini-batch documents. XTR estimates the similarity score between query and document based on the retrieved top-k set only as follows:

$$f(Q, D) = \frac{1}{Z} \sum_{i}^{n} \max_{j \in |D|} \hat{A}_{ij} q_i^T d_j \qquad (1)$$

Here, the normalizer Z denotes the count of query tokens that retrieve at least one document token from D. During training, the cross-entropy loss over in-batch negatives is used, expressed as:

$$\mathcal{L}_{CE} = -\log \frac{\exp f(Q, D^+))}{\sum_{b=1}^{B} \exp f(Q, D_b))} \qquad (2)$$

Such a training enables XTR to retrieve documents based on top-k matching document tokens per query token, giving 400x speed up (Lee et al., 2023) against ColBERT which, instead, uses all document tokens for scoring.

4.2 UR^2N Background: Mono-T5

UR^2N in built on Mono-T5 which is T5 model fine-tuned for reranking. Mono-T5 first encodes each query-document pair, and the decoder then generates the relevance label for ranking as follows:

$$Query: Q \;\; Document: D \;\; Relevant: \qquad (3)$$

where Q and D are the query and document texts, respectively. Mono-T5 is fine-tuned to generate the words "true" or "false", where the probability of generation can be used as a relevance score to rerank documents

4.3 UR^2N Architecture

Figure 1 depicts the architecture of UR^2N, being built over Mono-T5 as reranker and extending it further for retrieval. To design UR^2N, we clone the last layer of Mono-T5 encoder to introduce a parallel last layer exclusively for XTR retrieval task. Instead of the whole encoder, we only do targeted finetuning of this last parallel layer using XTR similarity function from equation 1, keeping

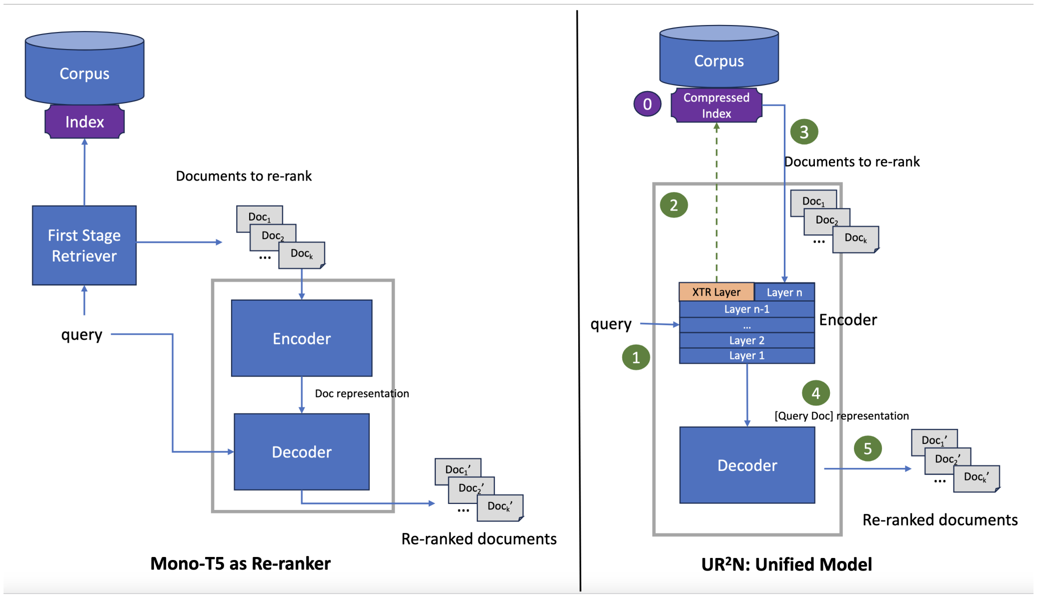

Figure 1: UR^2N Overview of UR^2N. Mono-T5 as Re-ranker has two stages: a first-stage retriever is used to get top-k documents for re-ranking. In UR^2N we add a parallel layer (XTR Layer) to the last layer of the encoder, initialized with the parameters of the last layer. During retrieval, we start with a pre-computed index (0). The given query (1) passes through the encoder layers (including XTR layer) to produce a query representation (2), which is used to retrieve top-k documents (3) for ranking. The query and the documents (4) are passed through the Mono-T5's encoder and decoder to obtain rankings for the documents (5).

rest of the encoder layers frozen. Specifically the XTR layer can be expressed as:

$$Y_{XTR} = LayerNorm(X_l + MultiHead(Y_l)) \tag{4}$$

where X_l is the input to the last layer, $MultiHead$ performs the attention mechanism, and $LayerNorm$ adds layer normalization. This design choice allows us tune the encoder for XTR style retrieval with only a small set of parameters, while retaining most of encoder params unchanged, enabling UR^2N to reap the benefits of encoder-decoder coupling learnt during Mono-T5 reranking. To reduce space footprint, we follow Colbert and add a linear layer to compress the encoder embeddings from XTR layer to lower dimensions ($dim = 128$)[1]. The transformation is expressed as:

$$E = Linear(Y_{XTR}) = Y_{XTR} \cdot W^T \tag{5}$$

The embeddings E are computed at token level for a document and query to be used in inner products, and subsequently the loss as per the equations 1 and 2.

[1]We show the index optimization impact in Appendix B

4.4 Retrieve and rerank in UR^2N

For a query Q, the token level embeddings E produced from T5 encoder with the XTR layer are used to search for nearest top-k document token embeddings. Benefiting from XTR style retriever tuning, during inference we consider only those documents as retrieved documents where at least one token from the document appeared in top-k retrieved tokens, given any query token.

For reranking, we use the pretrained Mono-T5 encoder-decoder as it is without the XTR layer for reranking the documents retrieved in the 1st stage.

5 Experiments

5.1 Setup and baselines

We evaluate UR^2N as a unified model for retrieval and reranking. We purposefully limit our experiments with UR^2N to *base* and *large* variation of the models trained with 1 GPU only to stress on its practical applicability with resource constraints.[2] We evaluate UR^2N on two research questions:
RQ1: Can UR^2N as a unified single model for retriever+reranker match the accuracy of traditional two stage multi-model retriever+reranker?

[2]We provide more implementation details and the exact set of hyper parameters in Appendix A

RQ2: Can UR^2N as a unified single model for retriever+reranker be better in retrieval than standalone retrievers?

To have a comprehensive comparison we use 4 baselines as (1) Mono-T5 as Reranker (2) BM25 as Retriever (3) ColBERT-v2 as Retriever (4) XTR as Retriever.

5.2 Benchmark and Evaluation Metric

We use the subset of 13 datasets from BIER (Thakur et al., 2021) benchmark which was used to benchmark XTR (Lee et al., 2023) as our evaluation benchmark. Similar to XTR, we train UR^2N on MS-Marco and do zero shot evaluations on the mentioned subset of BIER. Following BIER benchmark standard, we use Normalised Discounted Cumulative Gain (NDCG) (Wang et al., 2013) as our evaluation metric. For both retriever and reranker use cases, we compute NDCG@10.

5.3 Results

In this section, we review the experimental results in detail for both use-cases. We begin with the reranker use-case.

5.3.1 Use-case I: UR^2N as Reranker

In table 1 we compare UR^2N as a reranker which reranks the documents retrieved by its own encoder tuned in XTR style, with Mono-t5 which reranks documents retrieved by BM25. For both systems, we have reranked top 100 documents. We can see both Mono-T5 and UR^2N does better at NDCG@10 as compared to BM25, establishing the utility of a reranker. More specifically, we see UR^2N gains almost 7.5% on average across 13 datasets from BIER as compared to BM25 and almost matches the performance of a state-of-the-art reranker Mono-T5. This is significant as it answers our first research question RQ1 in affirmative: We can indeed unify retrieval and reranking into a single model and still retain the state-of-the-art reranker accuracy.

Table 2 shows same comparisons but with the large variant of the models. With large models UR^2N in fact does better than Mono-T5 by almost 1.5% on average over 13 datasets. This is because with large models, the encoder representations in UR^2N has more scope of learning robust representations to retrieve a better set of documents to rerank. This improvement also validates another important point that unified modeling with neural retrievers can offer significant improvements with larger sized models than using BM25 like static retrievers in retrieve and rerank pipeline.

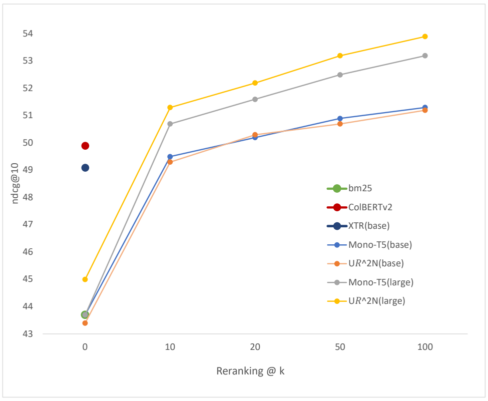

Figure 2: Plot of NDCG@10 against varying number of retrieved documents (k).

To investigate the role of number of retrieved documents, Figure 2 shows the graph of how the NDCG@10 numbers evolve after reranking, as we vary k across both systems including their base and large variations. We also show the retriever number from BM25, ColBERT-v2, XTR (base) to help the viewer understand the accuracy map of rerankers compared to retrievers.

As we see in Figure 2 reranker performance improves for both Mono-T5 and UR^2N as we increase k. In fact, with only k=*10* retrieved documents to rerank, both systems catch up with XTR (base) in terms of retriever performance. The reranker numbers improve more as we increase k to 100.

Comparing their performance across base and large variant of models again shows that UR^2N gains much more with large variants than Mono-T5. Mono-T5 (large) numbers are almost identical with Mono-T5 (base) across different values of k, implying Mono-T5 is not able to leverage any benefit from a higher model capacity with BM25 as a static 1st stage retriever. This further confirms our hypothesis that having a neural model as first stage retriever helps in scaling up performance with model capacity as seen in UR^2N.

Figure 3 further zooms into the performance difference between base and large model variations exclusively for UR^2N. As we see here, UR^2N gains a lot by increasing the model capacity from base to large and that is seen even for first stage retrieval (marked as without reranker in the figure) numbers. Because UR^2N has better first stage retrieval

Datasets	AR	TO	FE	CF	SF	CV	NF	NQ	HQ	FQ	SD	DB	QU	Avg.
BM25	39.7	44.0	65.1	0.17	67.9	59.5	32.2	0.31	0.63	23.6	14.9	31.8	78.9	43.7
Mono-T5(base)	35.9	30.6	81.2	24.7	73.7	78.5	35.7	52.4	71.2	39.3	16.7	42.9	84.1	51.3
UR^2N(base)	38.3	26.6	82.1	24.1	73.7	80.5	35.2	55.0	69.5	40.8	17.1	39.4	83.3	51.2

Table 1: Comparing Reranker performance for UR^2N:base models

Datasets	AR	TO	FE	CF	SF	CV	NF	NQ	HQ	FQ	SD	DB	QU	Avg.
BM25	39.7	44.0	65.1	0.17	67.9	59.5	32.2	0.31	0.63	23.6	14.9	31.8	78.9	43.7
Mono-T5(large)	44.5	30.2	82.7	25.3	74.5	81.5	36.4	55.6	72.7	42.6	18.5	42.6	85.0	53.2
UR^2N(large)	46.1	29.8	83.7	25.5	75.2	81.2	37.4	58.8	71.5	46.0	19.0	41.9	84.2	53.8

Table 2: Comparing Reranker performance for UR^2N:large models

Datasets	AR	TO	FE	CF	SF	CV	NF	NQ	HQ	FQ	SD	DB	QU	Avg.
BM25	39.7	44.0	65.1	0.17	67.9	59.5	32.2	0.31	0.63	23.6	14.9	31.8	78.9	43.7
ColBERT v2	46.3	26.3	78.5	17.6	69.3	73.8	33.8	56.2	66.7	35.6	15.4	44.6	85.2	49.9
XTR (base)	40.7	31.3	73.7	20.7	71.0	73.6	34.0	53.0	64.7	34.7	14.5	40.9	86.1	49.1
UR^2N(base)	38.3	26.6	82.1	24.1	73.7	80.5	35.2	55.0	69.5	40.8	17.1	39.4	83.3	51.2

Table 3: UR^2N as Retriever

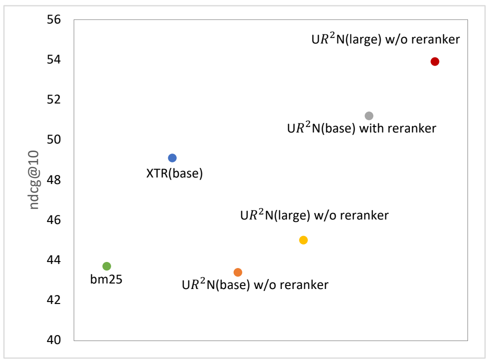

Figure 3: Comparing retriever and reranker with base and large variants

with large models, it easily opens up the space for reranker improvements too, which finally helps UR^2N with reranker beat XTR numbers.

5.3.2 Use-case II: UR^2N as Retriever

The unified modeling of UR^2N enables an end user to treat it as a black-box which receives a query and internally performs retrieval and reranking using the same model and finally provides top-k documents from the corpus similar to how a retriever I/O works. Therefore, in this section we treat the result of UR^2N as an end-to-end retriever and compare it with relevant baselines as shown in Table 3.

Table 3 shows dense models ColBERT-v2 and XTR as expected does better than BM25 as a retriever. However, the most important result from Table 3 is UR^2N as an end-to-end retriever performs the best among all. This is significant considering the fact that both ColBERT-v2 and XTR are very strong baselines owing to their multi-vector embedding-based approach. To design UR^2N, we adopted XTR style training with compressed representations to make UR^2N lightweight. Thus UR^2N performing better than ColBERT-v2 and XTR not only validates our design choices but also answers RQ2 (from sec 5) in affirmative. The unified training regime in UR^2N indeed enables it to be used as an end-to-end retriever performing better than standalone strong retrievers.

6 Conclusion and Future Work

We have proposed a unified encoder-decoder based architecture UR^2N that can use its own encoder representations for first-stage retrieval, and the same encoder-decoder network for reranking the retrieved documents. We have taken the perspective of a practising R&D engineer with practical resource constraints to design UR^2N as a lightweight architecture on top of Mono-T5 with XTR style finetuning. We have empirically shown that UR^2N trained with all the practical constraints, provides competitive/better performance than state-of-the-art systems both as reranker and as an end-to-end retriever. We also show that the unified modelling of retrieval and reranking can scale much better with model capacity and increased resource allocations. We believe this can open up a lot of research avenues in pursuing unified modelling as a serious research direction too.

References

Deng Cai, Yan Wang, Wei Bi, Zhaopeng Tu, Xiaojiang Liu, and Shuming Shi. 2019. Retrieval-guided dialogue response generation via a matching-to-generation framework. In *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 1866–1875, Hong Kong, China. Association for Computational Linguistics.

Wei-Cheng Chang, X Yu Felix, Yin-Wen Chang, Yiming Yang, and Sanjiv Kumar. 2019. Pre-training tasks for embedding-based large-scale retrieval. In *International Conference on Learning Representations*.

Chroma. 2024. Chroma vector db. `https://www.trychroma.com/`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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. Association for Computational Linguistics.

Bettina Fazzinga and Thomas Lukasiewicz. 2010. Semantic search on the web. *Semantic Web*, 1(1-2):89–96.

Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2021. Splade v2: Sparse lexical and expansion model for information retrieval. *Preprint*, arXiv:2109.10086.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777, Toronto, Canada. Association for Computational Linguistics.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *International conference on machine learning*, pages 3929–3938. PMLR.

Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised dense information retrieval with contrastive learning. *Preprint*, arXiv:2112.09118.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781.

Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pages 39–48.

Jinhyuk Lee, Zhuyun Dai, Sai Meher Karthik Duddu, Tao Lei, Iftekhar Naim, Ming-Wei Chang, and Vincent Y Zhao. 2023. Rethinking the role of token retrieval in multi-vector retrieval. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. 2021. Sparse, dense, and attentional representations for text retrieval. *Transactions of the Association for Computational Linguistics*, 9:329–345.

Rodrigo Nogueira, Zhiying Jiang, Ronak Pradeep, and Jimmy Lin. 2020. Document ranking with a pre-trained sequence-to-sequence model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 708–718, Online. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

Stephen E. Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.

Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. ColBERTv2: Effective and efficient retrieval via lightweight late interaction. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3715–3734, Seattle, United States. Association for Computational Linguistics.

Yi Tay, Vinh Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, et al. 2022. Transformer memory as a differentiable search index. *Advances in Neural Information Processing Systems*, 35:21831–21843.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. Beir: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Ellen M. Voorhees and Dawn M. Tice. 2000. The TREC-8 question answering track. In *Proceedings of the Second International Conference on Language Resources and Evaluation (LREC'00)*, Athens, Greece. European Language Resources Association (ELRA).

Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2614–2627.

Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Tie-Yan Liu, and Wei Chen. 2013. A theoretical analysis of ndcg type ranking measures. *Journal of Machine Learning Research*, 30.

Wikipedia. 2024. Tf-idf score. `https://en.wikipedia.org/wiki/Tf%E2%80%93idf`.

Canwen Xu, Daya Guo, Nan Duan, and Julian McAuley. 2022. Laprador: Unsupervised pretrained dense retriever for zero-shot text retrieval. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 3557–3569.

Honglei Zhuang, Zhen Qin, Rolf Jagerman, Kai Hui, Ji Ma, Jing Lu, Jianmo Ni, Xuanhui Wang, and Michael Bendersky. 2023a. Rankt5: Fine-tuning t5 for text ranking with ranking losses. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 2308–2313, New York, NY, USA. Association for Computing Machinery.

Honglei Zhuang, Zhen Qin, Rolf Jagerman, Kai Hui, Ji Ma, Jing Lu, Jianmo Ni, Xuanhui Wang, and Michael Bendersky. 2023b. Rankt5: Fine-tuning t5 for text ranking with ranking losses. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 2308–2313, New York, NY, USA. Association for Computing Machinery.

A Implementation Details

We align our implementation in sync with the perspective we described in section 3.

We finetune the XTR-layer of both model sizes on MSMarco training set with a learning rate of 1-e3. XTR uses k_{train} parameter which we set to 52. Both models are trained on a single A100 80GB GPU, with a batch size of 24 for the base model and 52 for the large model. Moreover, we trained the models with hard negatives, one per positive query/document pair in a batch. The models were trained for 50K steps, and the best models based on the development set were used for the evaluation.

During XTR style retrieval, k=*number of document tokens* to retrieve, is varied depending on the size of the index. For smaller indexes, we set k to 500, while for larger ones, we increased it to 100,000. Note this k is at token retrieval an internal parameter of XTR, different from the "k" in top-k document retrieval for reranking.

B Optimized XTR index with ColBERT-v2 optimizations

Datasets	Faiss HNSW Flat Index(in GB)	ColBERT index(in GB)
NQ	860	25
NFCorpus	2.4	0.091
TREC COVID	67	3
Touché 2020	481	7

Table 4: Comparing the sizes of Faiss HNSW Flat indices and ColBERT indices

In Table 4, we give some empirical numbers to establish how the ColBERT-v2 optimizations we discussed in Section 4.4 helps in reducing the index size. Considering the first dataset NQ as an example, we can see it offers almost upto 97% shrinkage over the original index. On an average, we see the index size reduced by 98% across 4 datasets, which validates the need for our optimizations in designing UR^2N making the index management and deployment much cheaper and easier.