Efficient Nearest Neighbor Search for Cross-Encoder Models using Matrix Factorization

Nishant Yadav[†], **Nicholas Monath**[◇], **Rico Angell**[†], **Manzil Zaheer**[◇], and **Andrew McCallum**[†]
[†] University of Massachusetts Amherst [◇]Google Research
{nishantyadav, rangell, mccallum}@cs.umass.edu
{nmonath,manzilzaheer}@google.com

Abstract

Efficient k-nearest neighbor search is a fundamental task, foundational for many problems in NLP. When the similarity is measured by dot-product between dual-encoder vectors or ℓ_2-distance, there already exist many scalable and efficient search methods. But not so when similarity is measured by more accurate and expensive black-box neural similarity models, such as cross-encoders, which jointly encode the query and candidate neighbor. The cross-encoders' high computational cost typically limits their use to reranking candidates retrieved by a cheaper model, such as dual encoder or TF-IDF. However, the accuracy of such a two-stage approach is upper-bounded by the recall of the initial candidate set, and potentially requires additional training to align the auxiliary retrieval model with the cross-encoder model. In this paper, we present an approach that avoids the use of a dual-encoder for retrieval, relying solely on the cross-encoder. Retrieval is made efficient with CUR decomposition, a matrix decomposition approach that approximates all pairwise cross-encoder distances from a small subset of rows and columns of the distance matrix. Indexing items using our approach is computationally cheaper than training an auxiliary dual-encoder model through distillation. Empirically, for $k > 10$, our approach provides test-time recall-vs-computational cost trade-offs superior to the current widely-used methods that re-rank items retrieved using a dual-encoder or TF-IDF.

1 Introduction

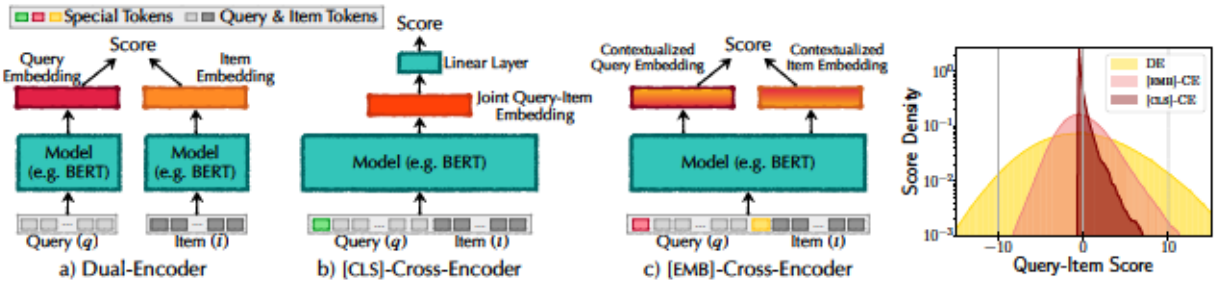
Finding top-k scoring items for a given query is a fundamental sub-routine of recommendation and information retrieval systems (Kowalski, 2007; Das et al., 2017). For instance, in question answering systems, the query corresponds to a question and the item corresponds to a document or a passage. Neural networks are widely used to model the similarity between a query and an item in such applications (Zamani et al., 2018; Hofstätter et al., 2019; Karpukhin et al., 2020; Qu et al., 2021). In this work, we focus on efficient k-nearest neighbor search for one such similarity function – the cross-encoder model.

Cross-encoder models output a scalar similarity score by jointly encoding the query-item pair and often generalize better to new domains and unseen data (Chen et al., 2020; Wu et al., 2020; Thakur et al., 2021) as compared to dual-encoder [1] models which independently embed the query and the item in a vector space, and use simple functions such as dot-product to measure similarity. However, due to the black-box nature of the cross-encoder based similarity function, the computational cost for brute force search with cross-encoders is prohibitively high. This often limits the use of cross-encoder models to re-ranking items retrieved using a separate retrieval model such as a dual-encoder or a TF-IDF-based model (Logeswaran et al., 2019; Zhang and Stratos, 2021; Qu et al., 2021). The accuracy of such a two-stage approach is upper bounded by the recall of relevant items by the initial retrieval model. Much of recent work either attempts to distill information from an expensive but more expressive cross-encoder model into a cheaper student model such as a dual-encoder (Wu et al., 2020; Hofstätter et al., 2020; Lu et al., 2020; Qu et al., 2021; Liu et al., 2022), or focuses on cheaper alternatives to the cross-encoder model while attempting to capture fine-grained interactions between the query and the item (Humeau et al., 2020; Khattab and Zaharia, 2020; Luan et al., 2021).

In this work, we tackle the fundamental task of efficient k-nearest neighbor search for a given query according to the cross-encoder. Our proposed approach, ANNCUR, uses CUR decomposition (Mahoney and Drineas, 2009), a matrix factorization approach, to approximate cross-encoder scores for all items, and retrieves k-nearest neighbor items while only making a small number of

[1] also referred to as two-tower models, Siamese networks

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 2171 - 2194
December 7-11, 2022



(a) Model architecture (b) Query-item score distribution

Figure 1: Model architecture and score distribution for three neural scoring functions. Dual-Encoders (DE) score a query-item pair using *independently* computed query and item embeddings. [CLS]-CE computes the score by jointly encoding the query-item pair followed by passing the joint query-item embedding through a linear layer. Our proposed [EMB]-CE embeds special tokens amongst query and item tokens, and computes the query-item score using query and item embeddings extracted using the special tokens after jointly encoding the query-item pair.

calls to the cross-encoder. Our proposed method selects a fixed set of anchor queries and anchor items, and uses scores between anchor queries and all items to generate latent embeddings for indexing the item set. At test time, we generate latent embedding for the query using cross-encoder scores for the test query and anchor items, and use it to approximate scores of all items for the given query and/or retrieve top-k items according to the approximate scores. In contrast to distillation-based approaches, our proposed approach does not involve any additional compute-intensive training of a student model such as dual-encoder via distillation.

In general, the performance of a matrix factorization-based method depends on the rank of the matrix being factorized. In our case, the entries of the matrix are cross-encoder scores for query-item pairs. To further improve rank of the score matrix, and in-turn performance of the proposed matrix factorization based approach, we propose [EMB]-CE which uses a novel dot-product based scoring mechanism for cross-encoder models (see Figure 1a). In contrast to the widely used [CLS]-CE approach of pooling query-item representation into a single vector followed by scoring using a linear layer, [EMB]-CE produces a score matrix with a much lower rank while performing at par with [CLS]-CE on the downstream task.

We run extensive experiments with cross-encoder models trained for the downstream task of entity linking. The query and item in this case correspond to a mention of an entity in text and a document with an entity description respectively. For the task of retrieving k-nearest neighbors according to the cross-encoder, our proposed approach presents superior recall-vs-computational cost trade-offs over using dual-encoders trained via distillation as well as over unsupervised TF-IDF-based methods (§3.2). We also evaluate the proposed method for various indexing and test-time cost budgets as well as study the effect of various design choices in §3.3 and §3.4.

2 Matrix Factorization for Nearest Neighbor Search

2.1 Task Description and Background

Given a scoring function $f_\theta : \mathcal{Q} \times \mathcal{I} \to \mathbb{R}$ that maps a query-item pair to a scalar score, and a query $q \in \mathcal{Q}$, the k-nearest neighbor task is to retrieve top-k scoring items according to the given scoring function f_θ from a fixed item set $\mathcal{I}$.

In NLP, queries and items are typically represented as a sequence of tokens and the scoring function is typically parameterized using deep neural models such as transformers (Vaswani et al., 2017). There are two popular choices for the scoring function – the cross-encoder (CE) model, and the dual-encoder (DE) model. The CE model scores a given query-item pair by concatenating the query and the item using special tokens, passing them through a model (such as a transformer $\mathcal{T}$) to obtain representation for the input pair followed by computing the score using linear weights $w \in \mathbb{R}^d$.

$$f_\theta^{(\text{CE})}(q, i) = w^\top \texttt{pool}(\mathcal{T}(\texttt{concat}(q, i)))$$

While effective, computing a similarity between a query-item pair requires a full forward pass of the model, which is often quite computationally burdensome. As a result, previous work uses auxiliary retrieval models such as BM25 (Robertson et al., 1995) or a trained dual-encoder (DE) model to approximate the CE. The DE model *independently*

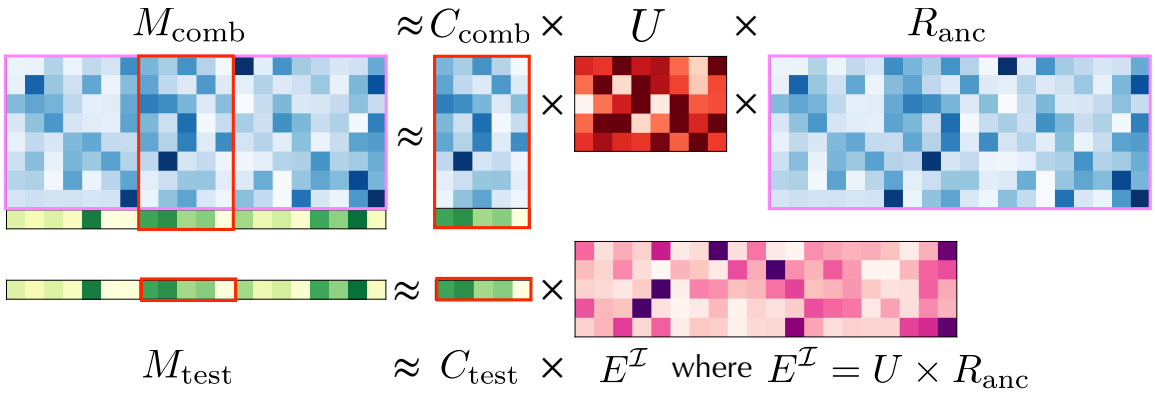


Figure 2: CUR decomposition of a matrix ($M_{\text{comb}} = \begin{bmatrix} M_{\text{anc}} \\ M_{\text{test}} \end{bmatrix}$) using a subset of its columns ($C_{\text{comb}} = \begin{bmatrix} C_{\text{anc}} \\ C_{\text{test}} \end{bmatrix}$) and rows ($R_{\text{anc}}$). The blue rows ($R_{\text{anc}}$) corresponding to the anchor queries and $U = C_{\text{anc}}^{\dagger}$ are used for indexing the items to obtain item embeddings $E^{\mathcal{I}} = U \times R_{\text{anc}}$ and the green row corresponds to the test query. Note that the test row (M_{test}) can be approximated using a subset of its columns (C_{test}) and the latent representation of items ($E^{\mathcal{I}}$).

embeds the query and the item in $\mathbb{R}^d$, for instance by using a transformer $\mathcal{T}$ followed by pooling the final layer representations into a single vector (e.g. using CLS token). The DE score for the query-item pair is computed using dot-product of the query embedding and the item embedding.

$$f_\theta^{(\text{DE})}(q, i) = \texttt{pool}(\mathcal{T}(q))^\top \texttt{pool}(\mathcal{T}(i))$$

In this work, we propose a method based on CUR matrix factorization that allows efficient retrieval of top-k items by directly approximating the cross-encoder model rather than using an auxiliary (trained) retrieval model.

CUR Decomposition (Mahoney and Drineas, 2009) In CUR matrix factorization, a matrix $M \in \mathbb{R}^{n \times m}$ is approximated using a subset of its rows $R = M[S_r, :] \in \mathbb{R}^{k_1 \times m}$, a subset of its columns $C = M[:, S_c] \in \mathbb{R}^{n \times k_2}$ and a joining matrix $U \in \mathbb{R}^{k_2 \times k_1}$ as follows

$$\tilde{M} = CUR$$

where S_r and S_c are the indices corresponding to rows R and columns C respectively, and the joining matrix U optimizes the approximation error. In this work, we set U to be the Moore-Penrose pseudo-inverse of $M[S_r, S_c]$, the intersection of matrices C and R, in which case $\tilde{M}$ is known as the skeleton approximation of M (Goreinov et al., 1997).

2.2 Proposed Method Overview

Our proposed method ANNCUR, which stands for **A**pproximate **N**earest **N**eighbor search using **CUR** decomposition, begins by selecting a fixed set of k_q *anchor* queries ($\mathcal{Q}_{\text{anc}}$) and k_i *anchor* items ($\mathcal{I}_{\text{anc}}$). It uses scores between queries $q \in \mathcal{Q}_{\text{anc}}$ and all items $i \in \mathcal{I}$ to index the item set by generating latent item embeddings. At test time, we compute exact scores between the test query $q_{\texttt{test}}$ and the anchor items $\mathcal{I}_{\text{anc}}$, and use it to approximate scores of all items for the given query and/or retrieve top-k items according to the approximate scores. We could optionally retrieve $k_r > k$ items, re-rank them using exact f_θ scores and return top-k items.

Let $M_{\text{comb}} = \begin{bmatrix} M_{\text{anc}} \\ M_{\text{test}} \end{bmatrix}$ and $C_{\text{comb}} = \begin{bmatrix} C_{\text{anc}} \\ C_{\text{test}} \end{bmatrix}$ where $M_{\text{anc}} = R_{\text{anc}} \in \mathbb{R}^{k_q \times |\mathcal{I}|}$ contains scores for the anchor queries and all items, $M_{\text{test}} \in \mathbb{R}^{1 \times |\mathcal{I}|}$ contains scores for a test query and all items, $C_{\text{anc}} \in \mathbb{R}^{k_q \times k_i}$ contains scores for the anchor queries and the anchor items, and $C_{\text{test}} \in \mathbb{R}^{1 \times k_i}$ contains scores for the test query paired with the anchor items.

Using CUR decomposition, we can approximate M_{comb} using a subset of its columns (C_{comb}) corresponding to the anchor items and a subset of its rows (R_{anc}) corresponding to the anchor queries as

$$\tilde{M}_{\text{comb}} = C_{\text{comb}} U R_{\text{anc}}$$

$$\begin{bmatrix} \tilde{M}_{\text{anc}} \\ \tilde{M}_{\text{test}} \end{bmatrix} = \begin{bmatrix} C_{\text{anc}} \\ C_{\text{test}} \end{bmatrix} U R_{\text{anc}}$$

$$\tilde{M}_{\text{anc}} = C_{\text{anc}} U R_{\text{anc}} \quad \text{and} \quad \tilde{M}_{\text{test}} = C_{\text{test}} U R_{\text{anc}}$$

Figure 2 shows CUR decomposition of matrix M_{comb}. At test time, $\tilde{M}_{\text{test}}$ containing approximate item scores for the test query can be computed using C_{test}, U, and R_{anc} where C_{test} contains exact f_θ scores between the test query and the anchor items. Matrices U and R_{anc} can be computed offline as these are independent of the test query.

2.3 Offline Indexing

The indexing process first computes R_{anc} containing scores between the anchor queries and all items.

$$R_{\text{anc}}(q, i) = f_\theta(q, i),\ \forall (q, i) \in \mathcal{Q}_{\text{anc}} \times \mathcal{I}$$

We embed all items in $\mathbb{R}^{k_i}$ as

$$E^{\mathcal{I}} = U \times R_{\text{anc}}$$

where $U = C_{\text{anc}}^{\dagger}$ is the pseudo-inverse of C_{anc}. Each column of $E^{\mathcal{I}} \in \mathbb{R}^{k_i \times |\mathcal{I}|}$ corresponds to a latent item embedding.

2.4 Test-time Inference

At test time, we embed the test query q in $\mathbb{R}^{k_i}$ using scores between q and anchor items $\mathcal{I}_{\text{anc}}$.

$$e_q = [f_\theta(q, i)]_{i \in \mathcal{I}_{\text{anc}}}$$

We approximate the score for a query-item pair (q, i) using inner-product of e_q and $E^{\mathcal{I}}[:, i]$ where $E^{\mathcal{I}}[:, i] \in \mathbb{R}^{k_i}$ is the embedding of item i.

$$\hat{f}_\theta(q, i) = e_q^\top E^{\mathcal{I}}[:, i]$$

We can use $e_q \in \mathbb{R}^{k_i}$ along with an off-the-shelf nearest-neighbor search method for maximum inner-product search (Malkov and Yashunin, 2018; Johnson et al., 2019; Guo et al., 2020) and retrieve top-scoring items for the given query q according to the approximate query-item scores *without* explicitly approximating scores for all the items.

2.5 Time Complexity

During indexing stage, we evaluate f_θ for $k_q|\mathcal{I}|$ query-item pairs, and compute the pseudo-inverse of a $k_q \times k_i$ matrix. The overall time complexity of the indexing stage is $\mathcal{O}(\mathcal{C}_{f_\theta} k_q |\mathcal{I}| + \mathcal{C}_{\text{inv}}^{k_q, k_i})$, where $\mathcal{C}_{\text{inv}}^{k_q, k_i}$ is the cost of computing the pseudo-inverse of a $k_q \times k_i$ matrix, and $\mathcal{C}_{f_\theta}$ is the cost of computing f_θ on a query-item pair. For CE models used in this work, we observe that $\mathcal{C}_{f_\theta} k_q |\mathcal{I}| \gg \mathcal{C}_{\text{inv}}^{k_q, k_i}$.

At test time, we need to compute f_θ for k_i query-item pairs followed by optionally re-ranking k_r items retrieved by maximum inner-product search (MIPS). Overall time complexity for inference is $\mathcal{O}\left((k_i + k_r)\mathcal{C}_{f_\theta} + \mathcal{C}_{\text{MIPS}}^{|\mathcal{I}|, k_r}\right)$, where $\mathcal{O}(\mathcal{C}_{\text{MIPS}}^{|\mathcal{I}|, k_r})$ is the time complexity of MIPS over $|\mathcal{I}|$ items to retrieve top-k_r items.

2.6 Improving score distribution of CE models for matrix factorization

The rank of the query-item score matrix, and in turn, the approximation error of a matrix factorization method depends on the scores in the matrix. Figure 1b shows a histogram of query-item score distribution (adjusted to have zero mean) for a dual-encoder and [CLS]-CE model. We use [CLS]-CE to refer to a cross-encoder model parameterized using transformers which uses CLS token to compute a pooled representation of the input query-item pair. Both the models are trained for zero-shot entity linking (see §3.1 for details). As shown in the figure, the query-item score distribution for the [CLS]-CE model is significantly skewed with only a small fraction of items (entities) getting high scores while the score distribution for a dual-encoder model is less so as it is generated explicitly using dot-product of query and item embeddings. The skewed score distribution from [CLS]-CE leads to a high rank query-item score matrix, which results in a large approximation error for matrix decomposition methods.

We propose a small but important change to the scoring mechanism of the cross-encoder so that it yields a less skewed score distribution, thus making it much easier to approximate the corresponding query-item score matrix *without* adversely affecting the downstream task performance. Instead of using CLS token representation to score a given query-item pair, we add special tokens amongst the query and the item tokens and extract *contextualized* query and item representations using the special tokens after *jointly* encoding the query-item pair using a model such as a transformer $\mathcal{T}$.

$$e_q^{\text{CE}}, e_i^{\text{CE}} = \texttt{pool}(\mathcal{T}(\texttt{concat}(q, i)))$$

The final score for the given query-item pair is computed using dot-product of the contextualized query and item embeddings.

$$f_\theta^{\text{([EMB]-CE)}}(q, i) = (e_q^{\text{CE}})^\top e_i^{\text{CE}}$$

We refer to this model as [EMB]-CE. Figure 1a shows high-level model architecture for dual-encoders, [CLS]-CE and [EMB]-CE model.

As shown in Figure 1b, the query-item score distribution from an [EMB]-CE model resembles that from a DE model. Empirically, we observe that rank of the query-item score matrix for [EMB]-CE model is much lower than the rank of a similar matrix computed using [CLS]-CE, thus making it much easier to approximate using matrix decomposition based methods.

3 Experiments

In our experiments, we use CE models trained for zero-shot entity linking on ZESHEL dataset (§3.1).

We evaluate the proposed method and various baselines on the task of finding k-nearest neighbors for cross-encoder models in §3.2, and evaluate the proposed method for various indexing and test-time cost budgets as well as study the effect of various design choices in §3.3 and §3.4. All resources for the paper including code for all experiments and model checkpoints is available at https://github.com/iesl/anncur

ZESHEL Dataset The **Ze**ro-**Sh**ot **E**ntity **L**inking (ZESHEL) dataset was constructed by Logeswaran et al. (2019) from Wikia. The task of zero-shot entity linking involves linking entity mentions in text to an entity from a list of entities with associated descriptions. The dataset consists of 16 different domains with eight, four, and four domains in training, dev, and test splits respectively. Each domain contains non-overlapping sets of entities, thus at test time, mentions need to be linked to unseen entities solely based on entity descriptions. Table 1 in the appendix shows dataset statistics. In this task, queries correspond to mentions of entities along with the surrounding context, and items correspond to entities with their associated descriptions.

3.1 Training DE and CE models on ZESHEL

Following the precedent set by recent papers (Wu et al., 2020; Zhang and Stratos, 2021), we first train a dual-encoder model on ZESHEL training data using hard negatives. We train a cross-encoder model for the task of zero-shot entity-linking on all eight training domains using cross-entropy loss with ground-truth entity and negative entities mined using the dual-encoder. We refer the reader to Appendix A.1 for more details.

Results on downstream task of Entity Linking To evaluate the cross-encoder models, we retrieve 64 entities for each test mention using the dual-encoder model and re-rank them using a cross-encoder model. The top-64 entities retrieved by the DE contain the ground-truth entity for 87.95% mentions in test data and 92.04% mentions in dev data. The proposed [EMB]-CE model achieves an average accuracy of 65.49 and 66.86 on domains in test and dev set respectively, and performs at par with the widely used and state-of-the-art [2] [CLS]-CE architecture which achieves an accuracy of 65.87 and 67.67 on test and dev set respectively. Since [EMB]-CE model performs at par with [CLS]-CE on the downstream task of entity linking, and rank of the score matrix from [EMB]-CE is much lower than that from [CLS]-CE, we use [EMB]-CE in subsequent experiments.

[2] We observe that our implementation of [CLS]-CE obtains slightly different results as compared to state-of-the-art (see Table 2 in Zhang and Stratos (2021)) likely due to minor implementation/training differences.

3.2 Evaluating on k-NN search for CE

Experimental Setup For all experiments in this section, we use the [EMB]-CE model trained on original ZESHEL training data on the task of zero-shot entity linking, and evaluate the proposed method and baselines for the task of retrieving k-nearest neighbor entities (items) for a given mention (query) according to the cross-encoder model.

We run experiments *separately* on five domains from ZESHEL containing 10K to 100K items. For each domain, we compute the query-item score matrix for a subset or all of the queries (mentions) and all items (entities) in the domain. We randomly split the query set into a training set ($\mathcal{Q}_{\text{train}}$) and a test set ($\mathcal{Q}_{\text{test}}$). We use the queries in training data to train baseline DE models. For ANNCUR, we use the training queries as anchor queries and use CE scores between the anchor queries and all items for indexing as described in §2.3. All approaches are then evaluated on the task of finding top-k CE items for queries in the corresponding domain's test split. For a fair comparison, we do not train DE models on multiple domains at the same time.

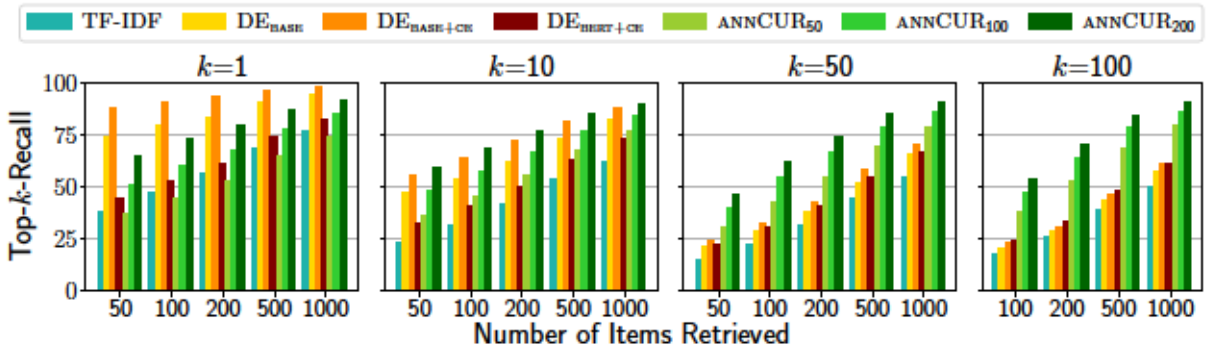
3.2.1 Baseline Retrieval Methods

TF-IDF: All queries and items are embedded using a TF-IDF vectorizer trained on item descriptions and top-k items are retrieved using the dot-product of query and item embeddings.

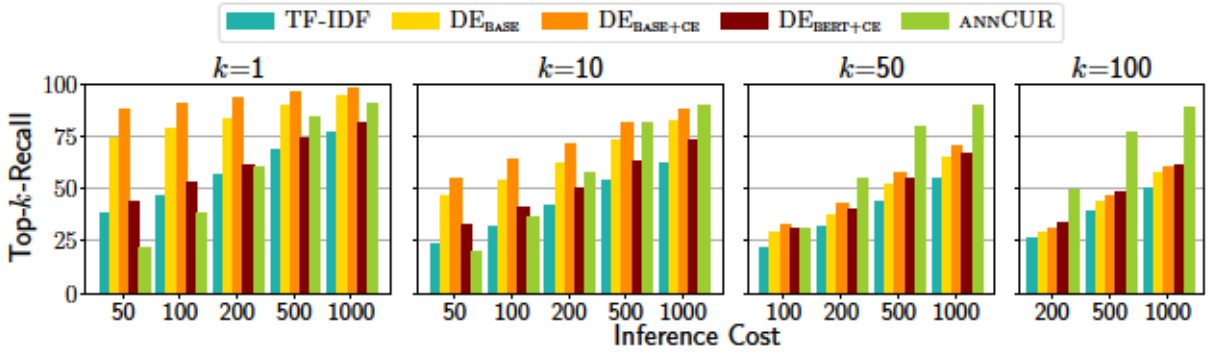
DE models: We experiment with DE_{BASE}, the DE model trained on ZESHEL for the task of entity linking (see §3.1), and the following two DE models trained via distillation from the CE.

- $\text{DE}_{\text{BERT+CE}}$: DE initialized with BERT (Devlin et al., 2019) and trained *only* using training signal from the cross-encoder model.
- $\text{DE}_{\text{BASE+CE}}$: DE_{BASE} model further fine-tuned via distillation using the cross-encoder model.

We refer the reader to Appendix A.3 for hyper-parameter and optimization details.



(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.



(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 3: Top-k-Recall results for domain=YuGiOh and $|\mathcal{Q}_{\text{train}}| = 500$

Evaluation metric We evaluate all approaches under the following two settings.

- In the first setting, we retrieve k_r items for a given query, re-rank them using exact CE scores and keep top-k items. We evaluate each method using Top-k-Recall@k_r which is the percentage of top-k items according to the CE model present in the k_r retrieved items.
- In the second setting, we operate under a fixed test-time cost budget where the cost is defined as the number of CE calls made during inference. Baselines such as DE and TF-IDF will use the entire cost budget for re-ranking items using exact CE scores while our proposed approach will have to split the budget between the number of anchor items (k_i) used for embedding the query (§2.4) and the number of items (k_r) retrieved for final re-ranking.

We refer to our proposed method as **ANNCUR**$_{k_i}$ when using fixed set of k_i anchor items chosen uniformly at random, and we refer to it as **ANNCUR** when operating under a fixed test-time cost budget in which case different values of k_i and k_r are used in each setting.

3.2.2 Results

Figures 3a and 3b show recall of top-k cross-encoder nearest neighbors for $k \in \{1, 10, 50, 100\}$ on ZESHEL domain = YuGiOh when using 500 queries for training, and evaluating on the remaining 2874 test queries. Figure 3a shows recall when each method retrieves the same number of items and Figure 3b shows recall when each method operates under a fixed inference cost budget.

Performance for $k \geq 10$ In Figure 3a, our proposed approach outperforms all baselines at finding top-$k = 10, 50$, and 100 nearest neighbors when all models retrieve the same number of items for re-ranking. In Figure 3b, when operating under the same cost budget, ANNCUR outperforms DE baselines at larger cost budgets for $k = 10, 50$, and 100. Recall that at a smaller cost budget, ANNCUR is able to retrieve fewer number of items for exact re-ranking than the baselines as it needs to use a fraction of the cost budget i.e. CE calls to compare the test-query with anchor items in order to embed the query for retrieving relevant items. Generally, the optimal budget split between the number of anchor items (k_i) and the number of items retrieved

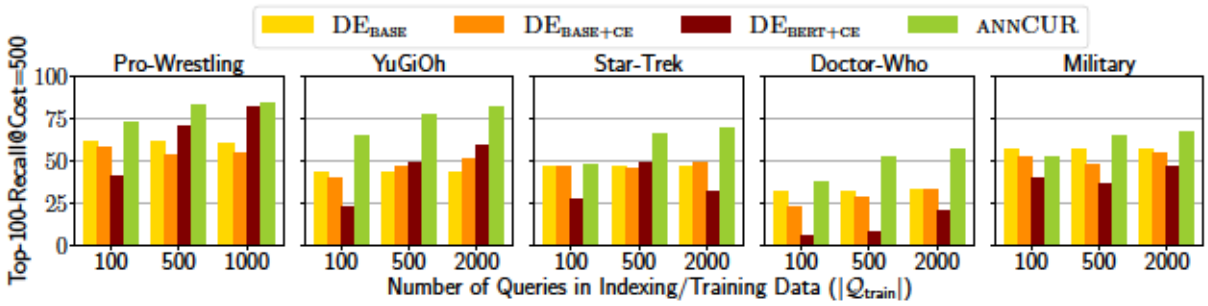


Figure 4: Bar plot showing Top-100-Recall@Cost=500 for different methods as we increase $|\mathcal{Q}_{\text{train}}|$, the size of indexing/training data for five different domains.

for exact re-ranking (k_r) allocates around 40-60% of the budget to k_i and the remaining budget to k_r.

Performance for $k = 1$ The top-1 nearest neighbor according to the CE is likely to be the ground-truth entity (item) for the given mention (query). Note that DE_{BASE} was trained using a massive amount of entity linking data (all eight training domains in ZESHEL, see §3.1) using the ground-truth entity (item) as the positive item. Thus, it is natural for top-1 nearest neighbor for both of these models to be aligned. For this reason, we observe that DE_{BASE} and $\text{DE}_{\text{BASE+CE}}$ outperform ANNCUR for $k = 1$. However, our proposed approach either outperforms or is competitive with $\text{DE}_{\text{BERT+CE}}$, a DE model trained *only* using CE scores for 500 queries after initializing with BERT. In Figure 3a, ANNCUR_{100} and ANNCUR_{200} outperform $\text{DE}_{\text{BERT+CE}}$ and in Figure 3b ANNCUR outperforms $\text{DE}_{\text{BERT+CE}}$ at larger cost budgets.

We refer the reader to Appendix B.3 for results on all combinations of top-k values, domains, and training data size values.

Effect of training data size ($|\mathcal{Q}_{\text{train}}|$) Figure 4 shows Top-100-Recall@Cost=500 on test queries for various methods as we increase the number of queries in training data ($|\mathcal{Q}_{\text{train}}|$). For DE baselines, the trend is not consistent across all domains. On YuGiOh, the performance consistently improves with $|\mathcal{Q}_{\text{train}}|$. However, on Military, the performance of distilled DE drops on going from 100 to 500 training queries but improves on going from 500 to 2000 training queries. Similarly, on Pro_Wrestling, performance of distilled $\text{DE}_{\text{BASE+CE}}$ does not consistently improve with training data size while it does for $\text{DE}_{\text{BERT+CE}}$. We suspect that this is due to a combination of various factors such as overfitting on training data, sub-optimal hyper-parameter configuration, divergence of model parameters etc. In contrast, our proposed method, ANNCUR, always shows consistent improvements as we increase the number of queries in training data, and avoids the perils of gradient-based training that often require large amounts of training data to avoid overfitting as well expensive hyper-parameter tuning in order to consistently work well across various domains.

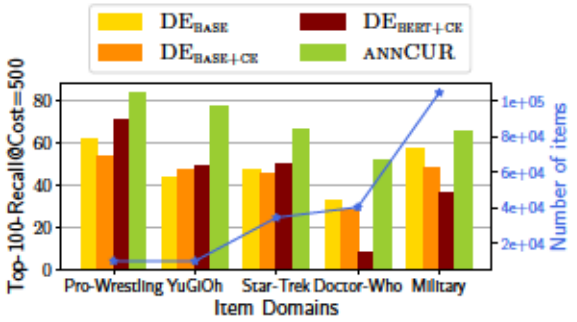
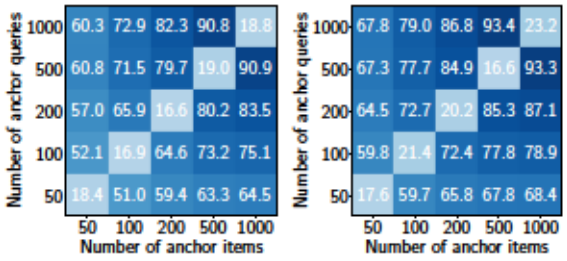


Figure 5: Bar plot with Top-100-Recall@Cost=500 for five domains in ZESHEL when using $|\mathcal{Q}_{\text{train}}| = 500$ queries for training/indexing and a line-plot showing the number of items (entities) in each domain.

Effect of domain size Figure 5 shows Top-100-Recall@Cost=500 for ANNCUR and DE baselines on primary y-axis and size of the domain i.e. total number of items on secondary y-axis for five different domains in ZESHEL. Generally, as the number of items in the domain increases, the performance of all methods drops.

Indexing Cost The indexing process starts by computing query-item CE scores for queries in train split. ANNCUR uses these scores for indexing the items (see §2.3) while DE baselines use these scores to find ground-truth top-k items for each query followed by training DE models using CE query-item scores. For domain=YuGiOh with 10031 items, and $|\mathcal{Q}_{\text{train}}| = 500$, the time taken to compute query-item scores for train/anchor queries ($t_{\text{CE-Mat}}$) $\approx$ 10 hours on an NVIDIA GeForce RTX2080Ti GPU/12GB memory, and



(a) [CLS]-CE (b) [EMB]-CE

Figure 6: Top-10-Recall@500 of ANNCUR for non-anchor queries on domain = YuGiOh for two cross-encoder models – [CLS]-CE and [EMB]-CE.

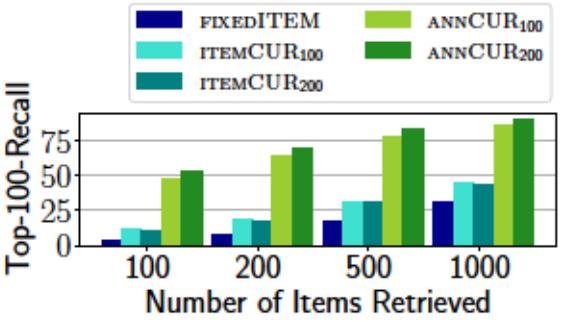


Figure 7: Bar plot showing Top-100-Recall for domain=YuGiOh when indexing using 500 anchor items for FIXEDITEM and ITEMCUR and 500 anchor queries for ANNCUR.

training a DE model further takes additional time ($t_{\text{DE-train}}$) $\approx$ 4.5 hours on two instances of the same GPU. Both $t_{\text{CE-Mat}}$ and $t_{\text{DE-train}}$ increase linearly with domain size and $|\mathcal{Q}_{\text{train}}|$, however the query-item score computation can be trivially parallelized. We ignore the time to build a nearest-neighbor search index over item embeddings produced by ANNCUR or DE as that is negligible in comparison to time spent on CE score computation and training of DE models. We refer the reader to Appendix A.3 for more details.

3.3 Analysis of ANNCUR

We compute the query-item score matrix for both [CLS]-CE and [EMB]-CE and compute the rank of these matrices using numpy (Harris et al., 2020) for domain=YuGiOh with 3374 queries (mentions) and 10031 items (entities). Rank of the score matrix for [CLS]-CE = 315 which is much higher than rank of the corresponding matrix for [EMB]-CE = 45 due to the query-item score distribution produced by [CLS]-CE model being much more skewed than that produced by [EMB]-CE model (see Fig. 1b).

Figures 6a and 6b show Top-10-Recall@500 on domain=YuGiOh for [CLS]-CE and [EMB]-CE respectively on different combinations of number of anchor queries (k_q) and anchor items (k_i). Both anchor queries and anchor items are chosen uniformly at random, and for a given set of anchor queries, we evaluate on the remaining set of queries.

[CLS]-CE versus [EMB]-CE For the same choice of anchor queries and anchor items, the proposed method performs better with [EMB]-CE model as compared [CLS]-CE due to the query-item score matrix for [EMB]-CE having much lower rank thus making it easier to approximate.

Effect of k_q and k_i Recall that the indexing time for ANNCUR is directly proportional to the number of anchor queries (k_q) while the number of anchor items (k_i) influences the test-time inference latency. Unsurprisingly, performance of ANNCUR increases as we increase k_i and k_q, and these can be tuned as per user's requirement to obtain desired recall-vs-indexing time and recall-vs-inference time trade-offs. We refer the reader to Appendix B.2 for a detailed explanation for the drop in performance when $k_q = k_i$.

3.4 Item-Item Similarity Baselines

We additionally compare with the following baselines that index items by comparing against a fixed set of anchor items [3] instead of anchor queries.

- FIXEDITEM: Embed all items and test-query in $\mathbb{R}^{k_i}$ using CE scores with a fixed set of k_i items chosen uniformly at random, and retrieve top-k_r items for the test query based on dot-product of these k_i-dim embeddings. We use $k_i = 500$.
- ITEMCUR-k_i: This is similar to the proposed approach except that it indexes the items by comparing them against k_i^{ind} anchor items instead of anchor queries for computing R_{anc} and C_{anc} matrices in the indexing step in §2.3. At test time, it performs inference just like ANNCUR (see §2.4) by comparing against a different set of fixed k_i anchor items. We use $k_i^{\text{ind}} = 500$.

Figure 7 shows Top-100-Recall for FIXEDITEM, ITEMCUR, and ANNCUR on domain = YuGiOh. ITEMCUR performs better than FIXEDITEM indicating that the latent item embeddings produced using CUR decomposition of the item-item similarity matrix are better than those built by comparing

[3] See appendix A.2 for details on computing item-item scores using a CE model trained to score query-item pairs.

the items against a fixed set of anchor items. ITEM-CUR performs worse than ANNCUR apparently because the CE was trained on query-item pairs and was not calibrated for item-item comparisons.

4 Related Work

Matrix Decomposition Classic matrix decomposition methods such as SVD, QR decomposition have been used for approximating kernel matrices and distance matrices (Musco and Woodruff, 2017; Tropp et al., 2017; Bakshi and Woodruff, 2018; Indyk et al., 2019). Interpolative decomposition methods such as Nyström method and CUR decomposition allow approximation of the matrix even when given only a subset of rows and columns of the matrix. Unsurprisingly, performance of these methods can be further improved if given the entire matrix as it allows for a better selection of rows and columns on the matrix used in the decomposition process (Goreinov et al., 1997; Drineas et al., 2005; Kumar et al., 2012; Wang and Zhang, 2013). Recent work, Ray et al. (2022) proposes sublinear Nyström approximations and considers CUR-based approaches for approximating non-PSD similarity matrices that arise in NLP tasks such as coreference resolution and document classification. Unlike previous work, our goal is to use the approximate scores to support retrieval of top scoring items. Although matrix decomposition methods for sparse matrices based on SVD (Berry, 1992; Keshavan et al., 2010; Hastie et al., 2015; Ramlatchan et al., 2018) can be used instead of CUR decomposition, such methods would require a) factorizing a sparse matrix at *test time* in order to obtain latent embeddings for all items and the test query, and b) indexing the latent item embeddings to efficiently retrieve top-scoring items for the given query. In this work, we use CUR decomposition as, unlike other sparse matrix decomposition methods, CUR decomposition allows for *offline* computation and indexing of item embeddings and the latent embedding for a test query is obtained simply by using its cross-encoder scores against the anchor items.

Cross-Encoders and Distillation Due to high computational costs, use of cross-encoders (CE) is often limited to either scoring a fixed set of items or re-ranking items retrieved by a separate (cheaper) retrieval model (Logeswaran et al., 2019; Qu et al., 2021; Bhattacharyya et al., 2021; Ayoola et al., 2022). CE models are also widely used for training computationally cheaper models via distillation on the training domain (Wu et al., 2020; Reddi et al., 2021), or for improving performance of these cheaper models on the target domain (Chen et al., 2020; Thakur et al., 2021) by using cross-encoders to score a fixed or heuristically retrieved set of items/datapoints. The DE baselines used in this work, in contrast, are trained using k-nearest neighbors for a given query according to the CE.

Nearest Neighbor Search For applications where the inputs are described as vectors in $\mathbb{R}^n$, nearest neighbor search has been widely studied for various (dis-)similarity functions such as ℓ_2 distance (Chávez et al., 2001; Hjaltason and Samet, 2003), inner-product (Jegou et al., 2010; Johnson et al., 2019; Guo et al., 2020), and Bregman-divergences (Cayton, 2008). Recent work on nearest neighbor search with non-metric (parametric) similarity functions explores various tree-based (Boytsov and Nyberg, 2019b) and graph-based nearest neighbor search indices (Boytsov and Nyberg, 2019a; Tan et al., 2020, 2021). In contrast, our approach approximates the scores of the parametric similarity function using the latent embeddings generated using CUR decomposition and uses off-the-shelf maximum inner product search methods with these latent embeddings to find k-nearest neighbors for the CE. An interesting avenue for future work would be to combine our approach with tree-based and graph-based approaches to further improve efficiency of these search methods.

5 Conclusion

In this paper, we proposed, ANNCUR, a matrix factorization-based approach for nearest neighbor search for a cross-encoder model without relying on an auxiliary model such as a dual-encoder for retrieval. ANNCUR approximates the test query's scores with all items by scoring the test-query only with a small number of anchor items, and retrieves items using the approximate scores. Empirically, for $k > 10$, our approach provides test-time recall-vs-computational cost trade-offs superior to the widely-used approach of using cross-encoders to re-rank items retrieved using a dual-encoder or a TF-IDF-based model. This work is a step towards enabling efficient retrieval with expensive similarity functions such as cross-encoders, and thus, moving beyond using such models merely for re-ranking items retrieved by auxiliary retrieval models such as dual-encoders and TF-IDF-based models.

Acknowledgements

We thank members of UMass IESL for helpful discussions and feedback. This work was supported in part by the Center for Data Science and the Center for Intelligent Information Retrieval, in part by the National Science Foundation under Grant No. NSF1763618, in part by the Chan Zuckerberg Initiative under the project "Scientific Knowledge Base Construction", in part by International Business Machines Corporation Cognitive Horizons Network agreement number W1668553, and in part using high performance computing equipment obtained under a grant from the Collaborative R&D Fund managed by the Massachusetts Technology Collaborative. Rico Angell was supported by the NSF Graduate Research Fellowship under Grant No. 1938059. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsor(s).

Limitations

In this work, we use cross-encoders parameterized using transformer models. Computing query-item scores using such models can be computationally expensive. For instance, on an NVIDIA GeForce RTX 2080Ti GPU with 12GB memory, we can achieve a throughput of approximately 140 scores/second, and computing a score matrix for 100 queries and 10K items takes about two hours. Although this computation can be trivially parallelized, the total amount of GPU hours required for this computation can be very high. However, note that these scores need to be computed even for distillation based DE baselines as we need to identify k-nearest neighbors for each query according to the cross-encoder model for training a dual-encoder model on this task.

Our proposed approach allows for indexing the item set only using scores from cross-encoder without any additional gradient based training but it is not immediately clear how it can benefit from data on multiple target domains at the same time. Parametric models such as dual-encoders on the other hand can benefit from training and knowledge distillation on multiple domains at the same time.

Ethical Consideration

Our proposed approach considers how to speed up the computation of nearest neighbor search for cross-encoder models. The cross-encoder model, which our approach approximates, may have certain biases / error tendencies. Our proposed approach does not attempt to mitigate those biases. It is not clear how those biases would propagate in our approximation, which we leave for future work. An informed user would scrutinize both the cross-encoder model and the resulting approximations used in this work.

References

Tom Ayoola, Shubhi Tyagi, Joseph Fisher, Christos Christodoulopoulos, and Andrea Pierleoni. 2022. ReFinED: An efficient zero-shot-capable approach to end-to-end entity linking. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Track*, pages 209–220. Association for Computational Linguistics.

Ainesh Bakshi and David Woodruff. 2018. Sublinear time low-rank approximation of distance matrices. *Advances in Neural Information Processing Systems*.

Michael W Berry. 1992. Large-scale sparse singular value computations. *The International Journal of Supercomputing Applications*, 6(1):13–49.

Sumanta Bhattacharyya, Amirmohammad Rooshenas, Subhajit Naskar, Simeng Sun, Mohit Iyyer, and Andrew McCallum. 2021. Energy-based reranking: Improving neural machine translation using energy-based models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4528–4537. Association for Computational Linguistics.

Leonid Boytsov and Eric Nyberg. 2019a. Accurate and fast retrieval for complex non-metric data via neighborhood graphs. In *International Conference on Similarity Search and Applications*, pages 128–142. Springer.

Leonid Boytsov and Eric Nyberg. 2019b. Pruning algorithms for low-dimensional non-metric k-nn search: a case study. In *International Conference on Similarity Search and Applications*, pages 72–85. Springer.

Lawrence Cayton. 2008. Fast nearest neighbor retrieval for bregman divergences. In *International Conference on Machine Learning*, pages 112–119.

Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. 2001. Searching in metric spaces. *ACM computing surveys (CSUR)*, 33(3):273–321.

Jiecao Chen, Liu Yang, Karthik Raman, Michael Bendersky, Jung-Jung Yeh, Yun Zhou, Marc Najork,

Danyang Cai, and Ehsan Emadzadeh. 2020. DiPair: Fast and accurate distillation for trillion-scale text matching and pair modeling. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 2925–2937. Association for Computational Linguistics.

Debashis Das, Laxman Sahoo, and Sujoy Datta. 2017. A survey on recommendation system. *International Journal of Computer Applications*, 160(7).

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. Association for Computational Linguistics.

Petros Drineas, Michael W Mahoney, and Nello Cristianini. 2005. On the nyström method for approximating a gram matrix for improved kernel-based learning. *The Journal of Machine Learning Research*, 6(12).

Sergei A Goreinov, Eugene E Tyrtyshnikov, and Nickolai L Zamarashkin. 1997. A theory of pseudoskeleton approximations. *Linear Algebra and its Applications*, 261(1-3):1–21.

Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*, pages 3887–3896.

Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. 2020. Array programming with NumPy. *Nature*, 585(7825):357–362.

Trevor Hastie, Rahul Mazumder, Jason D Lee, and Reza Zadeh. 2015. Matrix completion and low-rank svd via fast alternating least squares. *The Journal of Machine Learning Research*, 16(1):3367–3402.

Gisli R Hjaltason and Hanan Samet. 2003. Index-driven similarity search in metric spaces (survey article). *ACM Transactions on Database Systems (TODS)*, 28(4):517–580.

Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. 2020. Improving efficient neural ranking models with cross-architecture knowledge distillation. *ArXiv*, abs/2010.02666.

Sebastian Hofstätter, Navid Rekabsaz, Carsten Eickhoff, and Allan Hanbury. 2019. On the effect of low-frequency terms on neural-ir models. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1137–1140.

Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. 2020. Poly-encoders: Transformer architectures and pre-training strategies for fast and accurate multi-sentence scoring. In *International Conference on Learning Representations, ICLR*.

Pitor Indyk, Ali Vakilian, Tal Wagner, and David P Woodruff. 2019. Sample-optimal low-rank approximation of distance matrices. In *Conference on Learning Theory*, pages 1723–1751. PMLR.

Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128.

Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2019. Billion-scale similarity search with gpus. *IEEE Transactions on Big Data*, 7(3):535–547.

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. Association for Computational Linguistics.

Raghunandan H Keshavan, Andrea Montanari, and Sewoong Oh. 2010. Matrix completion from a few entries. *IEEE transactions on information theory*, 56(6):2980–2998.

Omar Khattab and Matei Zaharia. 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 39–48.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations, ICLR*.

Gerald J Kowalski. 2007. *Information retrieval systems: theory and implementation*, volume 1. Springer.

Sanjiv Kumar, Mehryar Mohri, and Ameet Talwalkar. 2012. Sampling methods for the nyström method. *The Journal of Machine Learning Research*, 13(1):981–1006.

Fangyu Liu, Yunlong Jiao, Jordan Massiah, Emine Yilmaz, and Serhii Havrylov. 2022. Trans-encoder: Unsupervised sentence-pair modelling through self-and mutual-distillations. In *International Conference on Learning Representations, ICLR*.

Lajanugen Logeswaran, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, Jacob Devlin, and Honglak Lee. 2019. Zero-shot entity linking by reading entity descriptions. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3449–3460. Association for Computational Linguistics.

Wenhao Lu, Jian Jiao, and Ruofei Zhang. 2020. Twinbert: Distilling knowledge to twin-structured compressed bert models for large-scale retrieval. In *ACM International Conference on Information & Knowledge Management*, pages 2645–2652.

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.

Michael W Mahoney and Petros Drineas. 2009. Cur matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences*, 106(3):697–702.

Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4):824–836.

Cameron Musco and David P Woodruff. 2017. Sublinear time low-rank approximation of positive semidefinite matrices. In *IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 672–683. IEEE.

Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. 2021. RocketQA: An optimized training approach to dense passage retrieval for open-domain question answering. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 5835–5847. Association for Computational Linguistics.

Andy Ramlatchan, Mengyun Yang, Quan Liu, Min Li, Jianxin Wang, and Yaohang Li. 2018. A survey of matrix completion methods for recommendation systems. *Big Data Mining and Analytics*, 1(4):308–323.

Archan Ray, Nicholas Monath, Andrew McCallum, and Cameron Musco. 2022. Sublinear time approximation of text similarity matrices. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(7):8072–8080.

Sashank Reddi, Rama Kumar Pasumarthi, Aditya Menon, Ankit Singh Rawat, Felix Yu, Seungyeon Kim, Andreas Veit, and Sanjiv Kumar. 2021. Rankdistil: Knowledge distillation for ranking. In *International Conference on Artificial Intelligence and Statistics*, pages 2368–2376. PMLR.

Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. 1995. Okapi at trec-3. *NIST Special Publication*, 109:109.

Shulong Tan, Weijie Zhao, and Ping Li. 2021. Fast neural ranking on bipartite graph indices. *Proceedings of the VLDB Endowment*, 15(4):794–803.

Shulong Tan, Zhixin Zhou, Zhaozhuo Xu, and Ping Li. 2020. Fast item ranking under neural network based measures. In *International Conference on Web Search and Data Mining*, pages 591–599.

Nandan Thakur, Nils Reimers, Johannes Daxenberger, and Iryna Gurevych. 2021. Augmented SBERT: Data augmentation method for improving bi-encoders for pairwise sentence scoring tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 296–310. Association for Computational Linguistics.

Joel A Tropp, Alp Yurtsever, Madeleine Udell, and Volkan Cevher. 2017. Randomized single-view algorithms for low-rank matrix approximation.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Shusen Wang and Zhihua Zhang. 2013. Improving cur matrix decomposition and the nyström approximation via adaptive sampling. *The Journal of Machine Learning Research*, 14(1):2729–2769.

Ledell Wu, Fabio Petroni, Martin Josifoski, Sebastian Riedel, and Luke Zettlemoyer. 2020. Scalable zero-shot entity linking with dense entity retrieval. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6397–6407. Association for Computational Linguistics.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Hamed Zamani, Mostafa Dehghani, W Bruce Croft, Erik Learned-Miller, and Jaap Kamps. 2018. From neural re-ranking to neural ranking: Learning a sparse representation for inverted indexing. In *ACM International Conference on Information and Knowledge Management*, pages 497–506.

Wenzheng Zhang and Karl Stratos. 2021. Understanding hard negatives in noise contrastive estimation. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1090–1101. Association for Computational Linguistics.

Domain	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{M}\rvert$	$\lvert\mathcal{M}_{k-\text{NN}}\rvert$
Training Data			
Military	104520	13063	2400
Pro Wrestling	10133	1392	1392
Doctor Who	40281	8334	4000
American Football	31929	3898	-
Fallout	16992	3286	-
Star Wars	87056	11824	-
World of Warcraft	27677	1437	-
Validation Data			
Coronation Street	17809	1464	-
Muppets	21344	2028	-
Ice Hockey	28684	2233	-
Elder Scrolls	21712	4275	-
Test Data			
Star Trek	34430	4227	4227
YuGiOh	10031	3374	3374
Forgotten Realms	15603	1200	-
Lego	10076	1199	-

Table 1: Statistics on number of entities $|\mathcal{E}|$ (items), total number of mentions $|\mathcal{M}|$ (queries), and number of mentions used in k-NN experiments ($|\mathcal{M}_{k-\text{NN}}|$) in §3.2 for each domain in ZESHEL dataset.

A Training Details

A.1 Training DE and CE for Entity Linking on ZESHEL

We initialize all models with `bert-base-uncased` and train using Adam (Kingma and Ba, 2015) optimizer with learning rate = 10^{-5}, and warm-up proportion=0.01 for four epochs. We evaluate on dev set five times during each epoch, and pick the model checkpoint that maximises accuracy on dev set. While training the dual-encoder model, we update negatives after each epoch using the latest dual-encoder model parameters to mine hard negatives. We trained the cross-encoder with a fixed set of 63 negatives items (entities) for each query (mention) mined using the dual-encoder model. We use batch size of 8 and 4 for training the dual-encoder and cross-encoder respectively.

Dual-encoder and cross-encoder models took 34 an 44 hours respectively for training on two NVIDIA GeForce RTX 8000 GPUs each with 48GB memory. The dual-encoder model has $2\times$110M parameters as it consists of separate query and item encoder models while the cross-encoder model has 110M parameters.

Tokenization details We use word-piece tokenization (Wu et al., 2016) for with a maximum of 128 tokens including special tokens for tokenizing entities and mentions. The mention representation consists of the word-piece tokens of the context surrounding the mention and the mention itself as

$$\texttt{[CLS]}\ \texttt{ctxt}_\texttt{l}\ \texttt{[M}_\texttt{s}\texttt{]}\ \texttt{ment}\ \texttt{[M}_\texttt{e}\texttt{]}\ \texttt{ctxt}_\texttt{r}\ \texttt{[SEP]}$$

where `ment`, $\texttt{ctxt}_\texttt{l}$, and $\texttt{ctxt}_\texttt{r}$ are word-piece tokens of the mention, context before and after the mention respectively, and $\texttt{[M}_\texttt{s}\texttt{]}$, $\texttt{[M}_\texttt{e}\texttt{]}$ are special tokens to tag the mention.

The entity representation is also composed of word-piece tokens of the entity title and description. The input to our entity model is:

$$\texttt{[CLS] title [ENT] description [SEP]}$$

where `title`, `description` are word-pieces tokens of entity title and description, and `[ENT]` is a special token to separate entity title and description representation.

The cross-encoder model takes as input the concatenated query (mention) and item (entity) representation with the `[CLS]` token stripped off the item (entity) tokenization as shown below

$$\texttt{[CLS]}\ \texttt{ctxt}_\texttt{l}\ \texttt{[M}_\texttt{s}\texttt{]}\ \texttt{ment}\ \texttt{[M}_\texttt{e}\texttt{]}\ \texttt{ctxt}_\texttt{r}\ \texttt{[SEP]}\ \ \texttt{title [ENT] description [SEP]}$$

A.2 Using query-item CE model for computing item-item similarity

We compute item-item similarity using a cross-encoder trained to score query-item pairs as follows. The query and item in our case correspond to mention of an entity with surrounding context and entity with its associated title and description respectively. We feed in first entity in the pair in the query slot by using mention span tokens around the title of the entity, and using entity description to fill in the right context of the mention. We feed in the second entity in the entity slot as usual. The concatenated representation of the entity pair (e_1, e_2) is given by

$$\texttt{[CLS]}\ \texttt{[M}_\texttt{s}\texttt{]}\ \texttt{t}_{\texttt{e}_1}\ \texttt{[M}_\texttt{e}\texttt{]}\ \texttt{d}_{\texttt{e}_1}\ \texttt{[SEP]}\ \texttt{t}_{\texttt{e}_2}\ \texttt{[E]}\ \texttt{d}_{\texttt{e}_2}\ \texttt{[SEP]}$$

where $\texttt{t}_{\texttt{e}_1}$, $\texttt{t}_{\texttt{e}_2}$ are the tokenized titles of the entities, $\texttt{d}_{\texttt{e}_1}$, $\texttt{d}_{\texttt{e}_2}$ are the tokenized entity descriptions, $\texttt{[M}_\texttt{e}\texttt{]}$, $\texttt{[M}_\texttt{s}\texttt{]}$ are special tokens denoting mention span boundary and `[E]` is a special token separating entity title from its description.

A.3 Training DE for k-NN retrieval with CE

We train dual-encoder models using k-nearest neighbor items according to cross-encoder model for each query using two loss functions. Let $S^{(\text{DE})}$ and $S^{(\text{CE})}$ be matrices containing score for all items for each query in training data. Let $\text{T}^{\text{CE}}_{k_d}(q), \text{T}^{\text{DE}}_{k_d}(q)$ be top-k_d items for query q according to the cross-encoder and dual-encoder respectively, and let $\text{N}^{\text{DE}}_{k_d}(q)$ be top-k_d items for query q according to dual-encoder that are not present in $\text{T}^{\text{CE}}_{k_d}(q)$.

We use loss functions $\mathcal{L}_{\text{match}}$ and $\mathcal{L}_{\text{pair}}$ described below for training the dual-encoder model using a cross-encoder model.

$$\mathcal{L}_{\text{match}} = \sum_{q \in \mathcal{Q}_{\text{train}}} \mathcal{H}\Big(\sigma(S^{(\text{DE})}_{[q,:]}), \sigma(S^{(\text{CE})}_{[q,:]})\Big)$$

where $\mathcal{H}$ is the cross-entropy function, and $\sigma(.)$ is the softmax function. In words, $\mathcal{L}_{\text{match}}$ is the cross-entropy loss between the dual-encoder and cross-encoder query-item score distribution over *all* items. Due to computational and memory limitations, we train by minimizing $\mathcal{L}_{\text{match}}$ using items in $\text{T}^{\text{CE}}_{k_d}(q)$ for each query $q \in \mathcal{Q}_{\text{train}}$.

$$\mathcal{L}_{\text{pair}} = \sum_{q \in \mathcal{Q}_{\text{train}}} \sum_{(i,j) \in \mathcal{P}_q} \mathcal{H}([0,1], \sigma([S^{(\text{DE})}_{q,i}, S^{(\text{DE})}_{q,j}]))$$

$$\text{where}, \mathcal{P}_q = \{(\text{T}^{\text{CE}}_{k_d}(q)_j, \text{N}^{\text{DE}}_{k_d}(q)_j)\}_{j=0}^{k_d}$$

$\mathcal{L}_{\text{pair}}$ treats items in $\text{T}^{\text{CE}}_{k_d}(q)$ as a positive item, pairs it with hard negatives from $\text{N}^{\text{DE}}_{k_d}(q)$, and minimizing $\mathcal{L}_{\text{pair}}$ increases dual-encoder's score for items in $\text{T}^{\text{CE}}_{k_d}(q)$, thus aligning $\text{T}^{\text{DE}}_{k_d}(q)$ with $\text{T}^{\text{CE}}_{k_d}(q)$ for queries in training data.

Training and optimization details We train all dual-encoder models using Adam optimizer with learning rate=10^{-5} for 10 epochs. We use a separate set of parameters for query and item encoders. We use 10% of training queries for validation and train on the remaining 90% of the queries. For each domain and training data size, we train with both $\mathcal{L}_{\text{match}}$ and $\mathcal{L}_{\text{pair}}$ loss functions, and pick the model that performs best on validation queries for k-NN retrieval according to the cross-encoder model.

We train models for with loss $\mathcal{L}_{\text{pair}}$ on two NVIDIA GeForce RTX 2080Ti GPUs with 12GB GPU memory and with loss $\mathcal{L}_{\text{match}}$ on two NVIDIA GeForce RTX 8000 GPUs with 48GB GPU memory as we could not train with $k_d = 100$ on 2080Tis due to GPU memory limitations. For loss $\mathcal{L}_{\text{pair}}$, we update the the list of negative items ($\text{N}^{\text{DE}}_{k_d}(q)$) for each query after each epoch by mining hard negative items using the latest dual-encoder model parameters.

$\lvert\mathcal{Q}_{\text{train}}\rvert$	Model	$t_{\text{CE-Mat}}$	t_{train}	t_{total}
100	DE-$\mathcal{L}_{\text{pair}}$	2	2.5	4.5
100	DE-$\mathcal{L}_{\text{match}}$	2	0.5	2.5
100	ANNCUR	2	-	2
500	DE-$\mathcal{L}_{\text{pair}}$	10	4.5	14.5
500	DE-$\mathcal{L}_{\text{match}}$	10	1	11
500	ANNCUR	10	-	10
2000	DE-$\mathcal{L}_{\text{pair}}$	40	11	51
2000	DE-$\mathcal{L}_{\text{match}}$	40	3	43
2000	ANNCUR	40	-	40

(a) Indexing time (in hrs) for ANNCUR and distillation based DE baselines for different number of anchor/train queries ($\lvert\mathcal{Q}_{\text{train}}\rvert$) for domain=YuGiOh.

Domain (w/ size)	Model	$t_{\text{CE-Mat}}$	t_{train}	t_{total}
YuGiOh-10K	DE-$\mathcal{L}_{\text{pair}}$	10	4.5	14.5
YuGiOh-10K	ANNCUR	10	-	10
Pro_Wrest-10K	DE-$\mathcal{L}_{\text{pair}}$	10	4.4	14.4
Pro_Wrest-10K	ANNCUR	10	-	10
Star_Trek-34K	DE-$\mathcal{L}_{\text{pair}}$	40	5.1	45.1
Star_Trek-34K	ANNCUR	40	-	40
Doctor_Who-40K	DE-$\mathcal{L}_{\text{pair}}$	40	5.2	45.2
Doctor_Who-40K	ANNCUR	40	-	40
Military-104K	DE-$\mathcal{L}_{\text{pair}}$	102	5.1	107.1
Military-104K	ANNCUR	102	-	102

(b) Indexing time (in hrs) for ANNCUR and distillation based DE baselines for various domains when using $\lvert\mathcal{Q}_{\text{train}}\rvert$=500 anchor/train queries.

Table 2: Indexing time breakdown for ANNCUR and DE models trained via distillation.

Indexing and Training Time Table 2a shows overall indexing time for the proposed method ANNCUR and for DE models trained using two distillation losses – $\mathcal{L}_{\text{pair}}$ and $\mathcal{L}_{\text{match}}$ on domain=YuGiOh. Training time (t_{train}) for loss $\mathcal{L}_{\text{match}}$ is much less as compared to that for $\mathcal{L}_{\text{pair}}$ as the former is trained on more powerful GPUs (two NVIDIA RTX8000s with 48GB memory each) due to its GPU memory requirements while the latter is trained on two NVIDIA 2080Ti GPUs with 12 GB memory each. The total indexing time (t_{total}) for DE models includes the time taken to compute CE score matrix ($t_{\text{CE-Mat}}$) because in order to train a DE model for the task of k-nearest neighbor search for a CE, we need to first find exact k-nearest neighbor items for the training queries. Note that this is different

Training Negatives	Dev Set [CLS]-CE	Dev Set [EMB]-CE	Test Set [CLS]-CE	Test Set [EMB]-CE
Random	59.60	57.74	58.72	56.56
TF-IDF	62.19	62.29	58.20	58.36
DE	67.67	66.86	65.87	65.49

(a) Macro-Average of Entity Linking Accuracy for [CLS]-CE and [EMB]-CE models on test and dev set in ZESHEL.

Training Negatives	[CLS]-CE	[EMB]-CE
Random	816	354
TF-IDF	396	67
DE	315	45

(b) Rank of 3374 × 10031 mention-entity cross-encoder score matrix for test domain = YuGiOh

Table 3: Accuracy on the downstream task of entity linking and rank of query-item (mention-entity) score matrix for [CLS]-CE and [EMB]-CE trained using different types of negatives.

from the "standard" way of training of DE models via distillation where the DE is often distilled using CE scores on a *fixed or heuristically retrieved* set of items, and *not* on k-nearest neighbor items according to the cross-encoder for a given query.

Table 2b shows indexing time for ANNCUR and DEs trained via distillation for five domains in ZESHEL. As the size of the domain increases, the time take for computing cross-encoder scores on training queries ($t_{\text{CE-Mat}}$) also increases. The time takes to train dual-encoder via distillation roughly remains the same as we train with fixed number of positive and negative items during distillation.

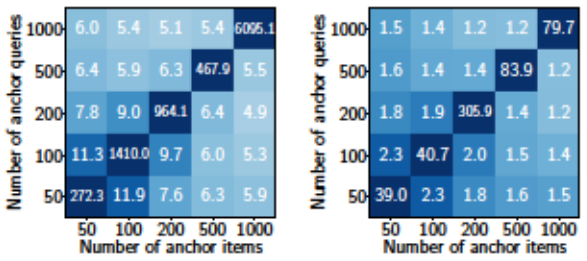
B Additional Results and Analysis

B.1 Comparing [EMB]-CE and [CLS]-CE

In addition to training cross-encoder models with negatives mined using a dual-encoder, we train both [CLS]-CE and [EMB]-CE models using random negatives and negatives mined using TF-IDF embeddings of mentions and entities. To evaluate the cross-encoder models, we retrieve 64 entities for each test mention using a dual-encoder model and re-rank them using a cross-encoder model.

Table 3a shows macro-averaged accuracy on the downstream task of entity linking over test and dev domains in ZESHEL dataset, and Table 3b shows rank of query-item score matrices on domain=YuGiOh for both cross-encoder models. The proposed [EMB]-CE model performs at par with the widely used [CLS]-CE architecture for all three kinds of negative mining strategies while producing a query-item score matrix with lower rank as compared to [CLS]-CE.

Figure 8 shows approximation error of ANNCUR for different combinations of number of anchor queries and anchor items for [CLS]-CE and [EMB]-CE. For a given set of anchor queries, the approximation error is evaluated on the remaining set of queries. The error between a matrix M and its approximation $\tilde{M}$ is measured as $\|M - \tilde{M}\|_F / \|M\|_F$ where $\|.\|_F$ is the Frobenius norm of a matrix. For the same choice of anchor queries and anchor items, the approximation error is lower for [EMB]-CE model as compared to [CLS]-CE. This aligns with the observation that rank of the query-item score matrix from [EMB]-CE is lower than the corresponding matrix from [CLS]-CE as shown in Table 3b.

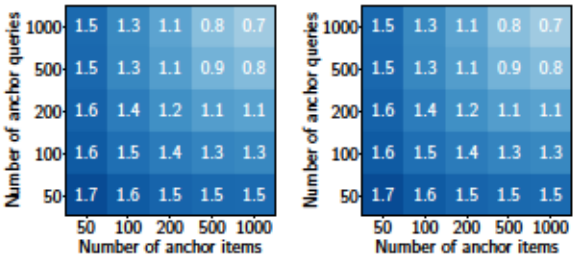


(a) [CLS]-CE (b) [EMB]-CE

Figure 8: Matrix approximation error evaluated on non-anchor queries for CUR decomposition on domain = YuGiOh for [CLS]-CE and [EMB]-CE models. The total number of queries including both anchor and non-anchor (test) queries is 3374 and the total number of items is 10031.

B.2 Understanding poor performance of ANNCUR for $k_i = k_q$

Figure 6 in §3.3 shows Top-10-Recall@500 on domain=YuGiOh for [CLS]-CE and [EMB]-CE respectively on different combinations of number of anchor queries (k_q) and anchor items (k_i). Note that the performance of ANNCUR drops significantly when $k_q = k_i$. Recall that the indexing step (§2.3) requires computing pseudo-inverse of matrix $A = M[\mathcal{Q}_{\text{anc}}, \mathcal{I}_{\text{anc}}]$ containing scores between anchor queries ($\mathcal{Q}_{\text{anc}}$) and anchor items ($\mathcal{I}_{\text{anc}}$). Performance drops significantly when A is a square matrix i.e. $k_q = k_i$ as the matrix tends to be ill-conditioned, with several very small eigenvalues that are 'blown up' in $A^{\dagger}$, the pseudo-inverse



(a) Top-10-Recall@Cost=500 (b) Matrix Approx. Error

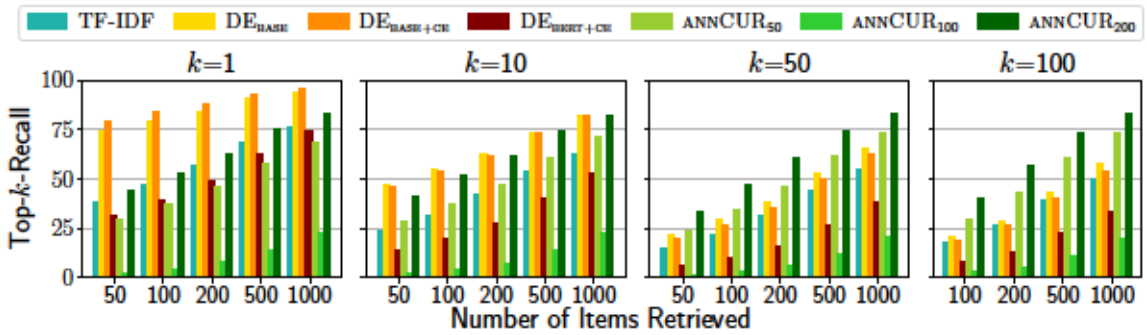
Figure 9: Performance of ANNCUR on non-anchor/test queries on domain = YuGiOh using $U = C^{\dagger}MR^{\dagger}$ for [EMB]-CE. The total number of queries including both anchor and non-anchor (test) queries is 3374 and the total number of items is 10031.

of A. This, in-turn, leads to a significant approximation error (Ray et al., 2022). Choosing different number of anchor queries and anchor items yields a rectangular matrix A whose eigenvalues are unlikely to be small, thus resulting in much better approximation of the matrix M.

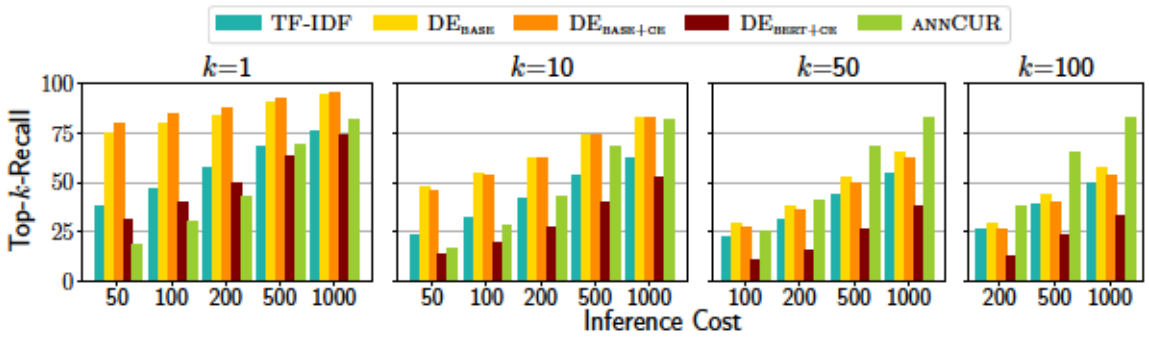
Oracle CUR Decomposition An alternate way of computing the matrix U in CUR decomposition of a matrix M for a given subset of rows (R) and columns (C) is to set $U = C^{\dagger}MR^{\dagger}$. This can provide a much stable approximation of the matrix M even when $k_q = k_i$ (Mahoney and Drineas, 2009). However, it requires computing *all* values of M before computing its low-rank approximation. In our case, we are trying to approximate a matrix M which also contains scores between test-queries and all items in order to avoid scoring all items using the CE model at test-time, thus we can not use $U = C^{\dagger}MR^{\dagger}$. Figure 9 shows results for an oracle experiment where we use $U = C^{\dagger}MR^{\dagger}$, and as expected it provides significant improvement when $k_q = k_i$ and minor improvement otherwise over using $U = M[\mathcal{Q}_{\text{anc}}, \mathcal{I}_{\text{anc}}]^{\dagger}$.

B.3 k-NN experiment results for all domains

For brevity, we show results for all top-k values only for domain=YuGiOh in the main paper. For the sake of completeness and for interested readers, we add results for combinations of top-k values, domains, and training data size values. Figure 10 - 24 contain results for top-$k \in \{1, 10, 50, 100\}$, for domain YuGiOh, Pro_Wrestling, Doctor_Who, Star_Trek, Military, and training data size $|\mathcal{Q}_{\text{train}}| \in \{100, 500, 2000\}$. For Pro_Wrestling, since the domain contains 1392 queries, we use maximum value of $|\mathcal{Q}_{\text{train}}| = 1000$ instead of 2000.

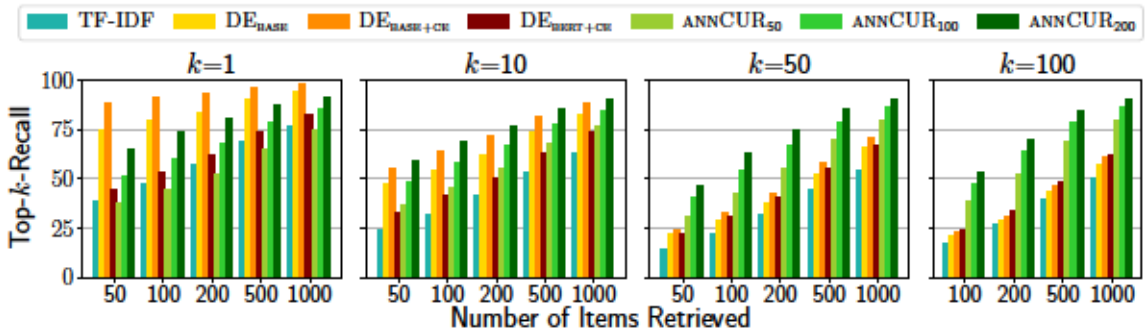


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

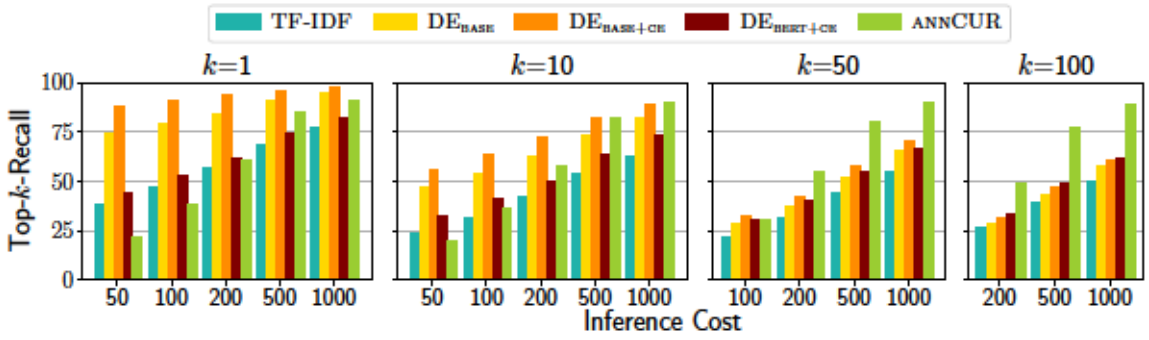


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 10: Top-k-Recall results for domain=YuGiOh and $|\mathcal{Q}_{\text{train}}| = 100$

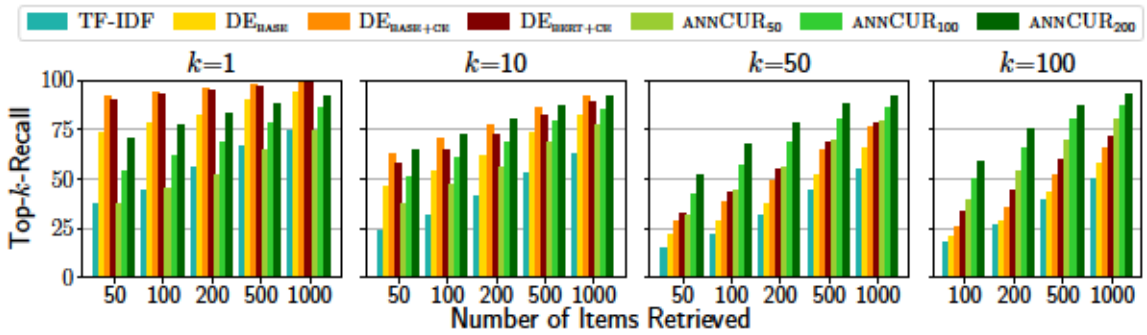


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

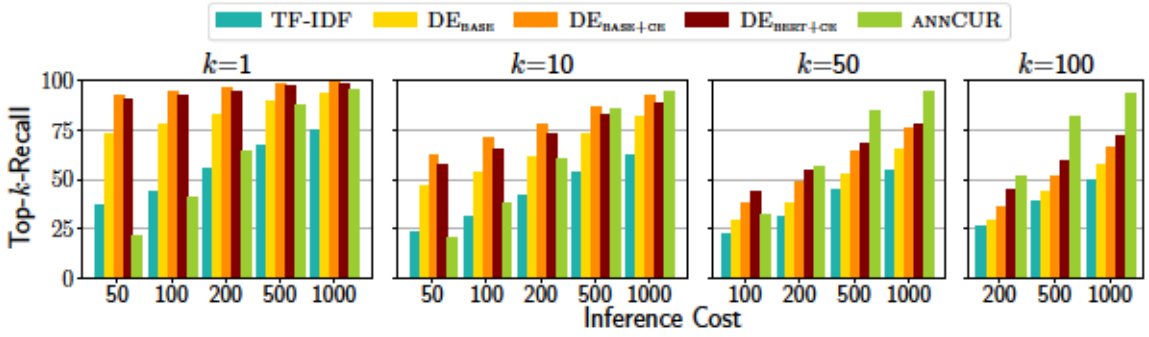


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 11: Top-k-Recall results for domain=YuGiOh and $|\mathcal{Q}_{\text{train}}| = 500$

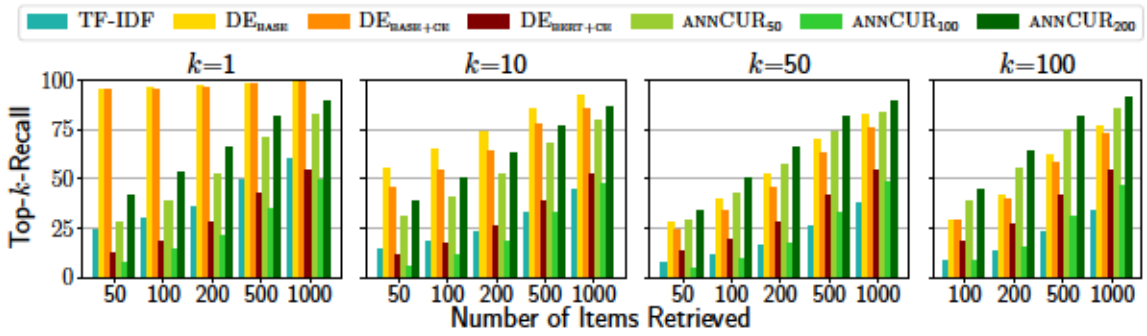


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

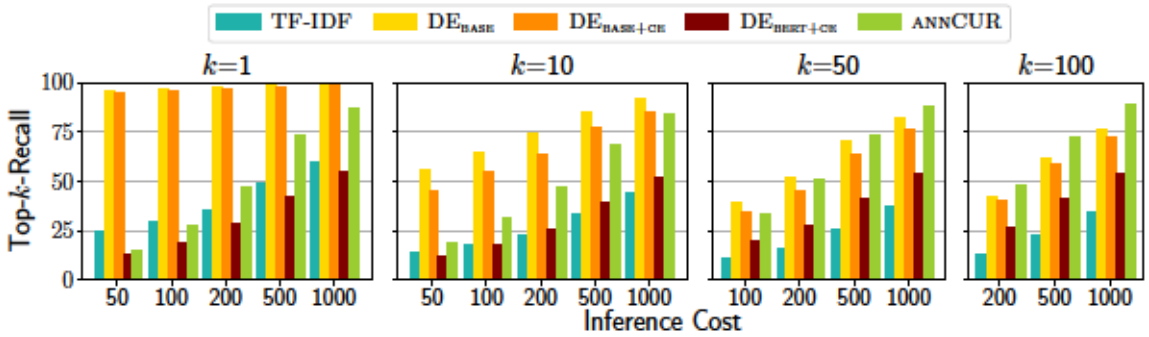


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 12: Top-k-Recall results for domain=YuGiOh and $|\mathcal{Q}_{\text{train}}| = 2000$

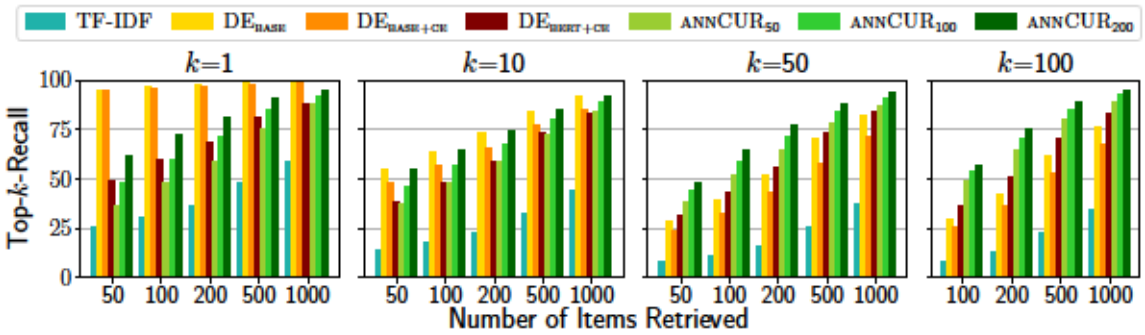


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

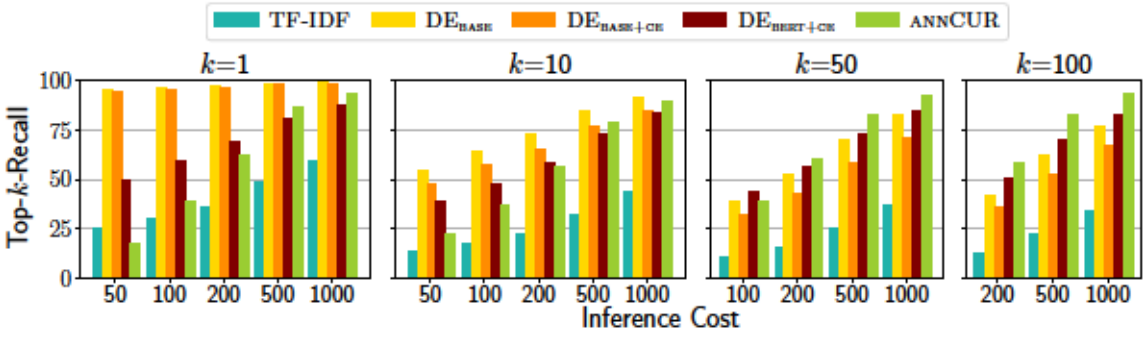


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 13: Top-k-Recall results for domain=Pro_Wrestling and $|\mathcal{Q}_{\text{train}}| = 100$

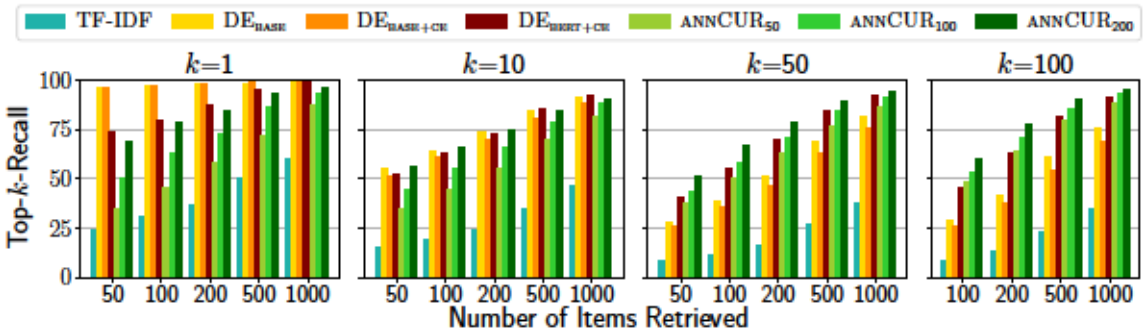


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

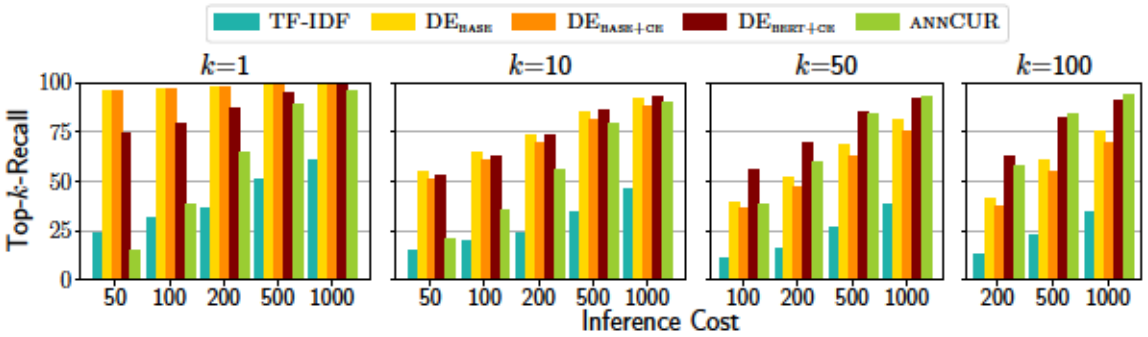


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 14: Top-k-Recall results for domain=Pro_Wrestling and $|\mathcal{Q}_{\text{train}}| = 500$

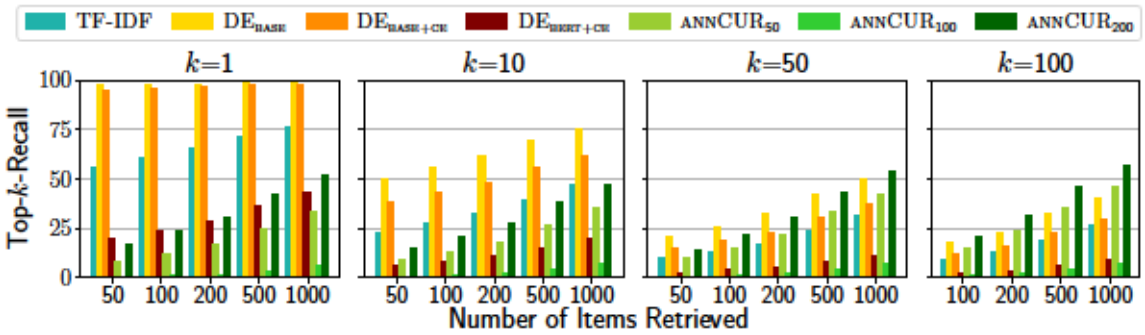


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

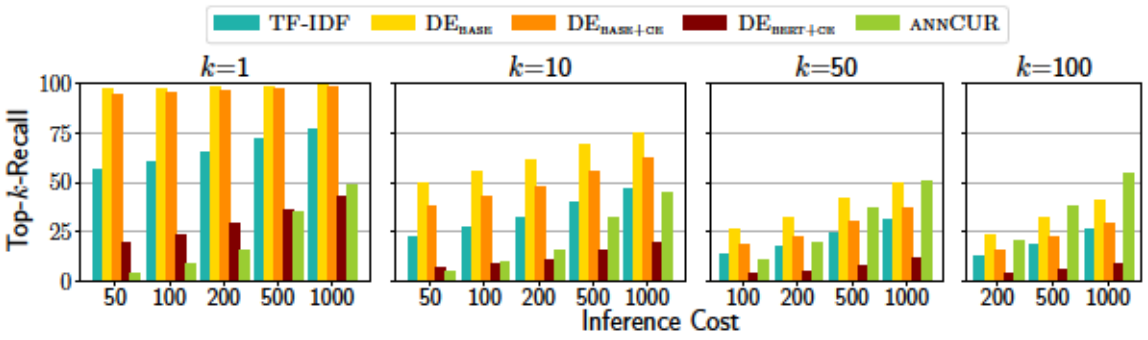


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 15: Top-k-Recall results for domain=Pro_Wrestling and $|\mathcal{Q}_{\text{train}}| = 1000$

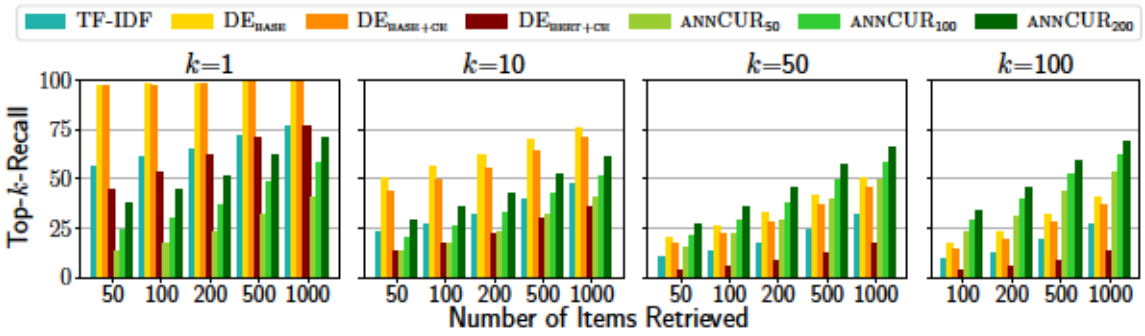


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

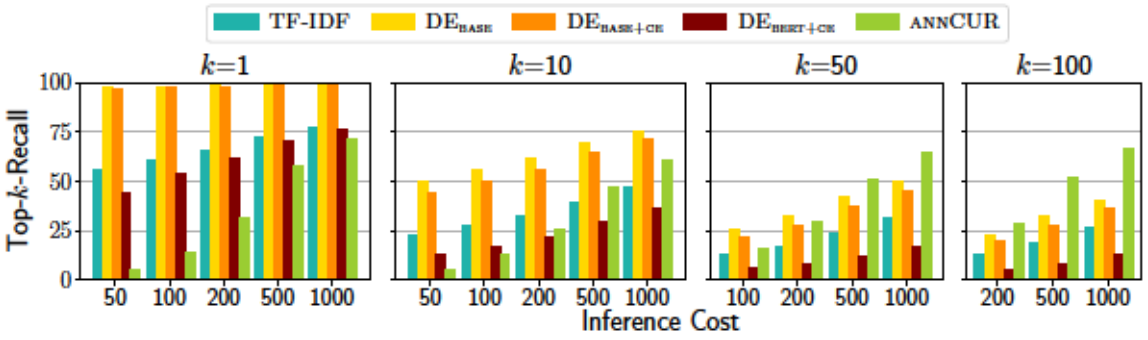


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 16: Top-k-Recall results for domain=Doctor_Who and $|\mathcal{Q}_{\text{train}}| = 100$

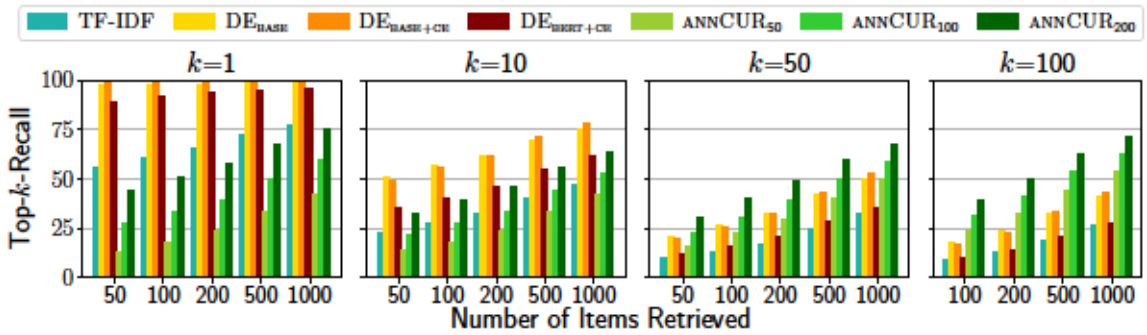


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

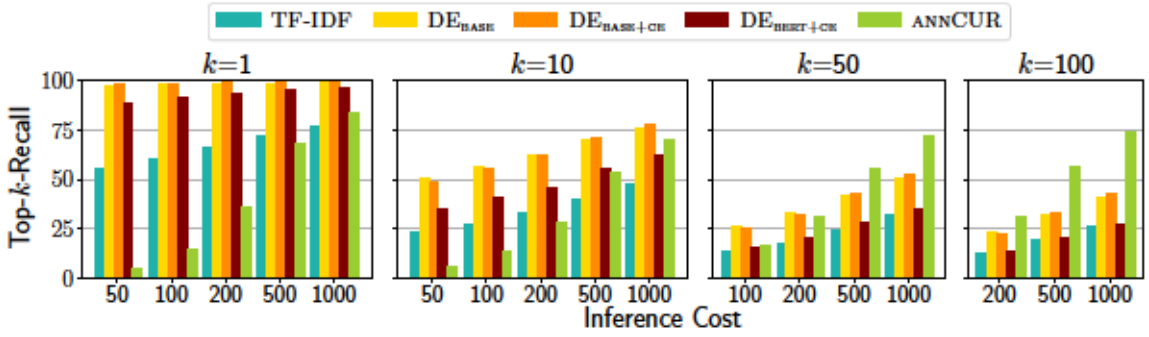


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 17: Top-k-Recall results for domain=Doctor_Who and $|\mathcal{Q}_{\text{train}}| = 500$

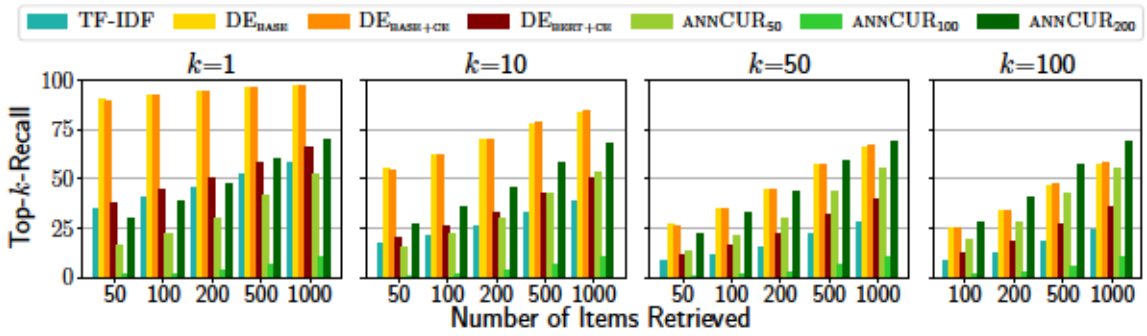


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

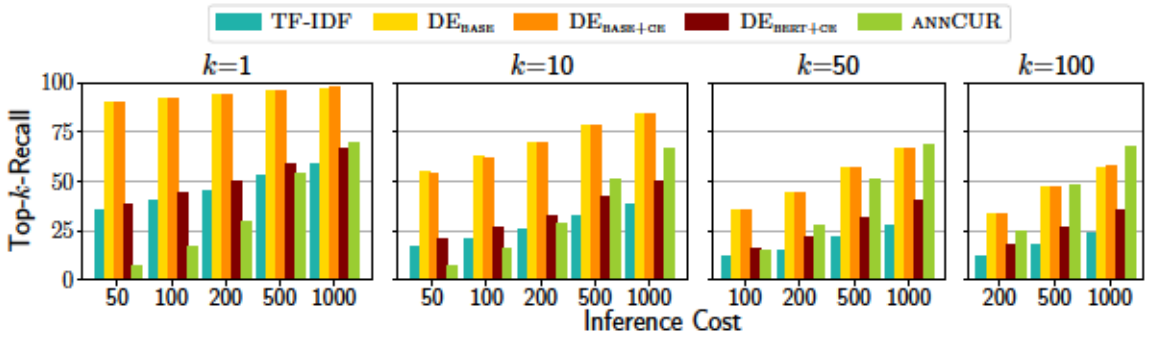


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 18: Top-k-Recall results for domain=Doctor_Who and $|\mathcal{Q}_{\text{train}}| = 2000$

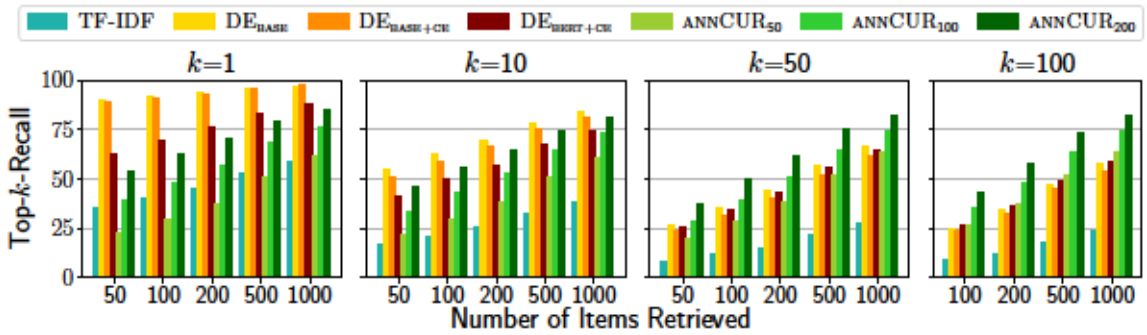


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

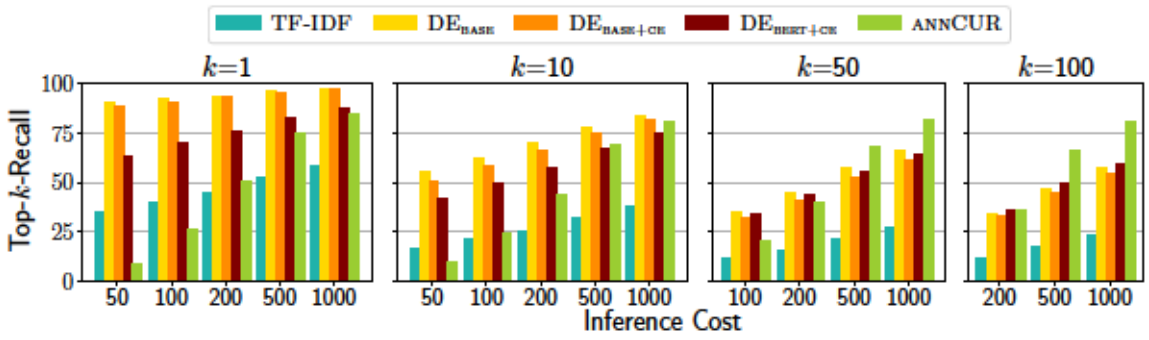


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 19: Top-k-Recall results for domain=Star_Trek and $|\mathcal{Q}_{\text{train}}| = 100$

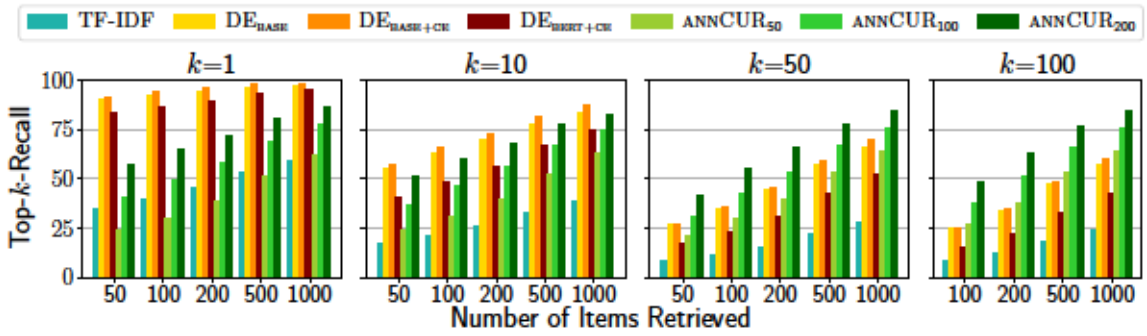


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

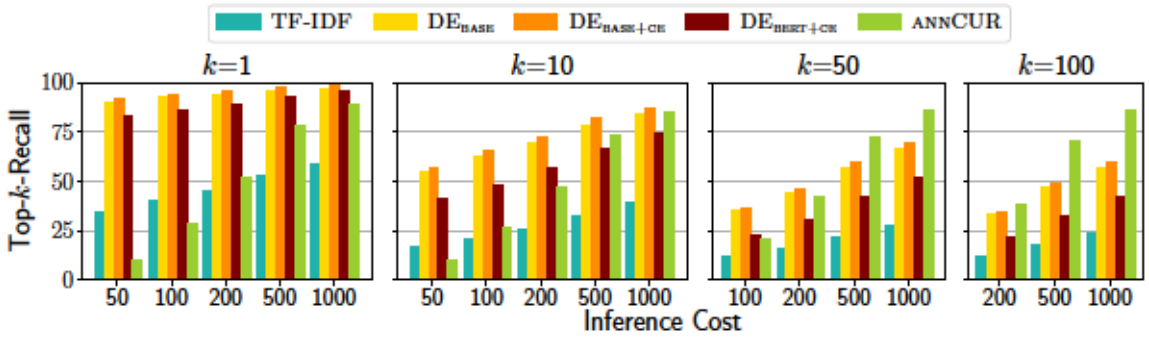


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 20: Top-k-Recall results for domain=Star_Trek and $|\mathcal{Q}_{\text{train}}| = 500$

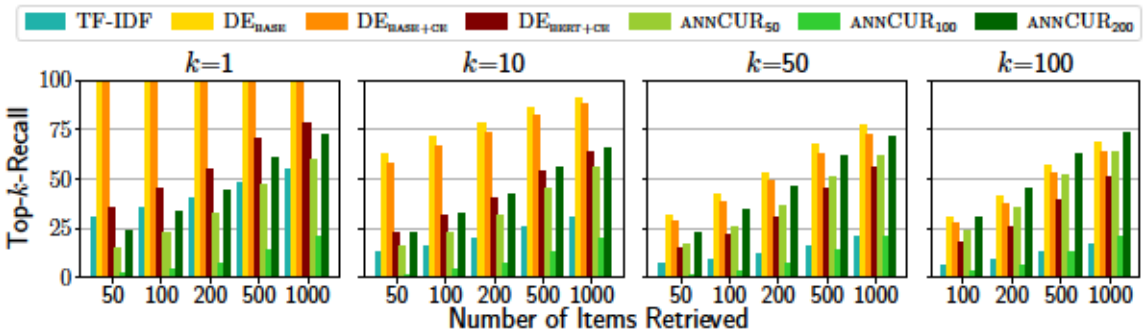


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

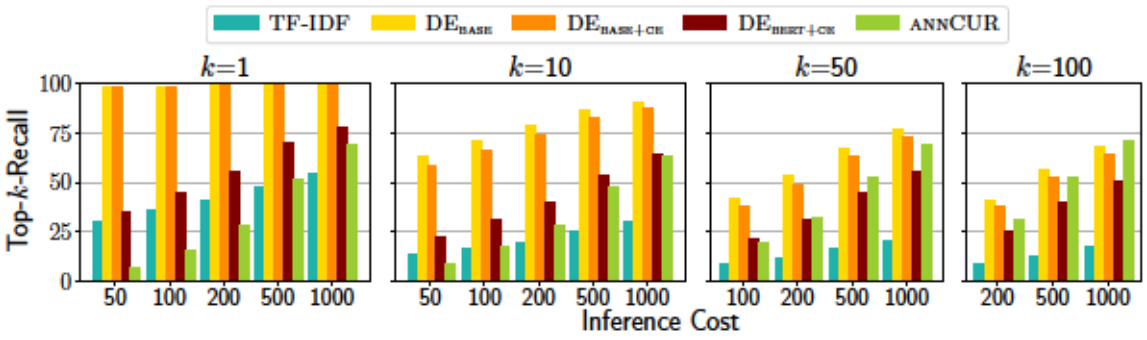


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 21: Top-k-Recall results for domain=Star_Trek and $|\mathcal{Q}_{\text{train}}| = 2000$

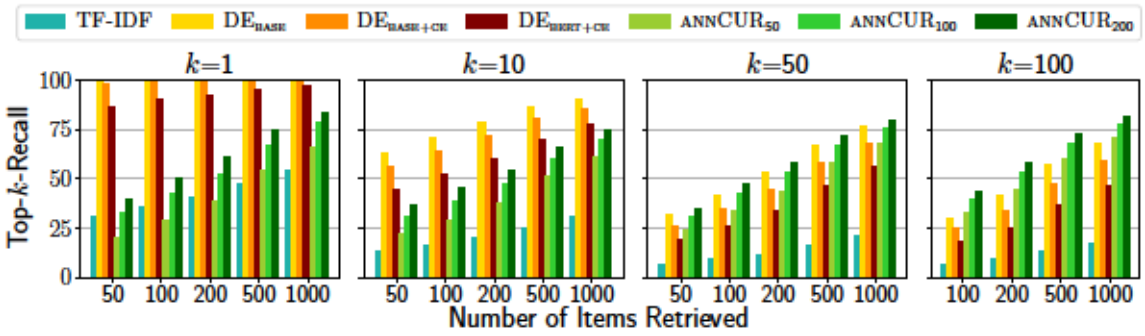


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

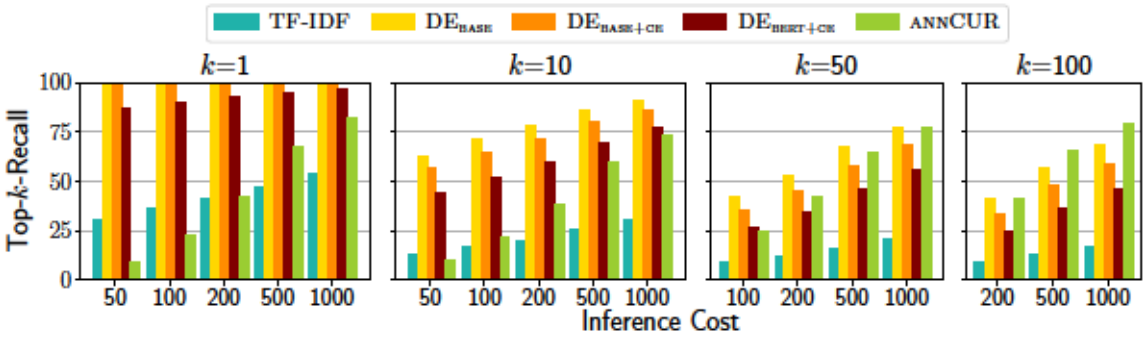


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 22: Top-k-Recall results for domain=`Military` and $|\mathcal{Q}_{\text{train}}| = 100$

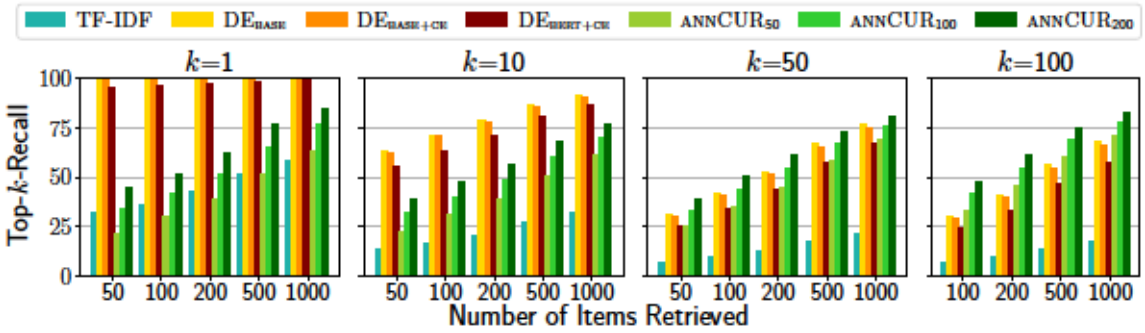


(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.

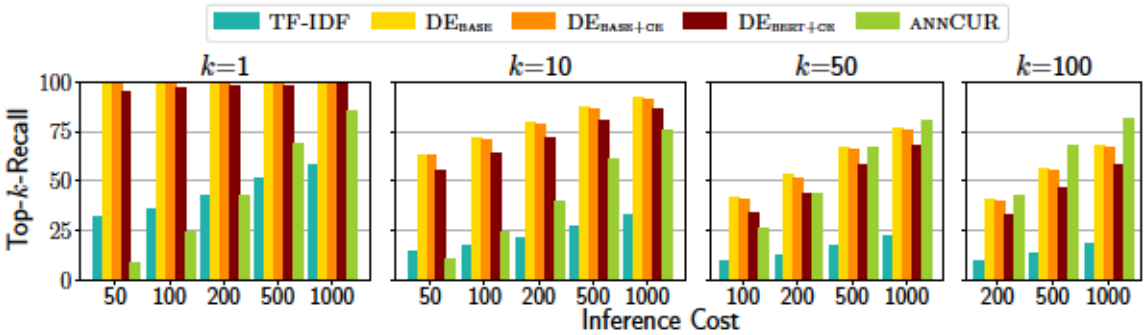


(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 23: Top-k-Recall results for domain=`Military` and $|\mathcal{Q}_{\text{train}}| = 500$



(a) Top-k-Recall@k_r for ANNCUR and baselines when all methods retrieve and rerank the same number of items (k_r). The subscript k_i in ANNCUR$_{k_i}$ refers to the number of anchor items used for embedding the test query.



(b) Top-k-Recall for ANNCUR and baselines when all methods operate under a fixed test-time cost budget. Recall that cost is the number of CE calls made during inference for re-ranking retrieved items and, in case of ANNCUR, it also includes CE calls to embed the test query by comparing with anchor items.

Figure 24: Top-k-Recall results for domain=`Military` and $|\mathcal{Q}_{\text{train}}| = 2000$