Once is Enough: A Light-Weight Cross-Attention for Fast Sentence Pair Modeling

Yuanhang Yang\textsuperscript{1}  Shiyi Qi\textsuperscript{1}  Cuiyun Gao\textsuperscript{1}  Zenglin Xu\textsuperscript{1}  Chuanyi Liu\textsuperscript{1}  Yulan He\textsuperscript{2}  Qifan Wang\textsuperscript{3}

Harbin Institute of Technology\textsuperscript{1}  University of Warwick\textsuperscript{2}  Meta AI\textsuperscript{3}

ysngkil@gmail.com, shiyiqi@stu.hit.edu.cn
{cuiyungao, zenglingxu, chuanyiliu}@hit.edu.cn
Yulan.He@warwick.ac.uk, wqfcr@fb.com

Abstract

Transformer-based models have achieved great success on sentence pair modeling tasks, such as answer selection and natural language inference (NLI). These models generally perform cross-attention over input pairs, leading to prohibitive computational cost. Recent studies propose dual-encoder and late interaction architectures for faster computation. However, the balance between the expressive of cross-attention and computation speedup still needs better coordinated. To this end, this paper introduces a novel paradigm MixEncoder for efficient sentence pair modeling. MixEncoder involves a light-weight cross-attention mechanism. It conducts query encoding only once while modeling the query-candidate interaction in parallel. Extensive experiments conducted on four tasks demonstrate that our MixEncoder can speed up sentence pairing by over 113x while achieving comparable performance as the more expensive cross-attention models.

1 Introduction

Transformer-based models (Vaswani et al., 2017; Devlin et al., 2019) have shown promising performance on sentence pair modeling tasks, such as natural language inference, question answering, information retrieval, etc (Nogueira and Cho, 2019; Qu et al., 2021; Zhao et al., 2021). Most pair modeling tasks can be depicted as a procedure of scoring the candidates given a query. A fundamental component of these models is the pre-trained cross-encoder, which models the interaction between the query and the candidates. As shown in Figure 1(a), the cross-encoder takes a pair of query and candidate as input, and calculates the interaction between them at each layer by the input-wide self-attention mechanism. This interaction will be calculated $N$ times if there are $N$ candidates. Despite the effective text representation power, it leads to exhaustive computation cost especially when the number of candidates is very large. This computation cost therefore restricts the use of these cross-encoder models in many real-world applications (Chen et al., 2020).

Extensive studies, including dual-encoder (Huang et al., 2013; Reimers and Gurevych, 2019) and late interaction models (MacAvaney et al., 2020; Gao et al., 2020; Chen et al., 2020; Khattab and Zaharia, 2020), have been proposed to accelerate the transformer inference on sentence pair modeling tasks. As shown in Figure 1(b), the query and candidates are processed separately in dual-encoders, thus the candidates can be pre-computed and cached for online inference, resulting in fast inference speed. However, this speedup is built upon sacrificing the expressiveness of cross-attention (Luan et al., 2021; Hu et al., 2021). Alternatively, late-interaction models adjust dual-encoders by appending an interaction component, such as a stack of Transformer layers (Cao et al., 2020; Nie et al., 2020), for modelling the interaction between the query and the cached candidates, as illustrated in Figure 1(c). Although this interaction components better preserve the effectiveness of cross-attention than dual-encoders, they still suffer from the heavy costs of the interaction component. Clearly, the computation cost of late-interaction models will still be dramatically increased as the number of candidates grows (Chen et al., 2020; Zhang et al., 2021).

To tackle the above issues, we propose a new paradigm named MixEncoder to speed up the inference while maintaining the expressiveness of cross-attention. In particular, MixEncoder involves a light-weight cross-attention mechanism which mostly disentangles the query encoding from query-candidate interaction. Specifically, MixEncoder encodes the query along with pre-computed candidates during runtime, and conducts the light-weight cross-attention at each interaction layer (named as interaction layer), as illustrated in Figure 1(d).
Figure 1: Architecture illustration of three popular sentence pair approaches and proposed MixEncoder, where \(N\) denotes the number of candidates and \(s\) denotes the relevance score of candidate-query pairs. The cache is used to store the pre-computed embeddings.

This design of light-weight cross-attention allows the interaction layer to process all the candidates in parallel. Thus, MixEncoder is able to encode the query only once, regardless of the number of candidates.

MixEncoder accelerates the online inference from two aspects. Firstly, MixEncoder processes each candidate into \(k\) dense context embeddings offline and cache them, where \(k\) is a hyper-parameter. This setup speeds up the online inference using pre-computed representations. Secondly, our interaction layer performs attention only from candidates to the query. This disentangles the query encoding from query-candidate interaction, thus avoiding repeatedly query encoding and supporting processing multiple candidates in parallel.

We evaluate the capability of MixEncoder for sentence pair modeling on four benchmark datasets, related to tasks of natural language inference, dialogue and information retrieval. The results demonstrate that MixEncoder better balances the effectiveness and efficiency. For example, MixEncoder achieves substantial speedup more than 113x over the cross-encoder and provides competitive performance.

In summary, our main contributions can be summarized as follows:

- A novel framework MixEncoder is proposed for fast and accurate sentence pair modeling. MixEncoder involves a light-weight cross-attention mechanism which allows us to encode the query once and process all the candidates in parallel.
- Extensive experiments on four public datasets demonstrate that the proposed MixEncoder provides better trade-offs between effectiveness and efficiency than state-of-the-art models.

2 Background and Related Work

Neural ranking models. These models focus on measuring the relevance of sentence pairs. A common practice is to map each sentence to a dense vector separately, and then measure their relevance with a similarity function (Huang et al., 2013; Karpukhin et al., 2020; Ren et al., 2021). These models are known as dual-encoder models. Dual-encoder models can pre-compute the candidate representations offline, since the candidate encoding is conducted independent of the query. Recently, pre-trained Transformer-based models (cross-encoder) have achieved great success on many sentence pair tasks (Li et al., 2022; Guo et al., 2022). These models take the concatenation of one sentence pair as input and perform cross-attention at each layer. This brings deep interactions between the input query and the candidate. Despite the promising performance, cross-encoder models will face significant latency in online inference since all the candidates are encoded online.

Late-interaction models. Various late interaction models have been proposed to combine the advantages of the dual-encoder and the cross-encoder. Specifically, these models disentangle the sentence pair modeling into separate encoding followed by a late interaction. They can pre-compute candidate
representations offline, and model the relationship of query-candidate pairs by cross-attention online.

For instance, the late-interaction models, including Deformer and PreTTR (MacAvaney et al., 2020), is based on a decomposed transformer, where low-level layers encode the query and candidate separately and the higher-level layers process them jointly. As shown in Figure 1(c), given $N$ candidates, the late Transformer layers have to encode the query $N$ times. It results in extensive computation costs. Other models propose to adopt a light-weight interaction mechanism, such as poly-attention (Humeau et al., 2020) and MaxSim (Khattab and Zaharia, 2020), instead of Transformer layers to speed up the online inference.

Our MixEncoder can behave as a late interaction model by replacing the upper Transformer layers of a dual-encoder with our interaction layer. The novelty of the MixEncoder lies in the light-weight cross-attention mechanism and pre-computed context embeddings.

3 Method

In this section, we first introduce the details of proposed MixEncoder, which mainly includes two stages, i.e., candidate pre-computation stage and query encoding stage. Figure 2 provides the architecture of MixEncoder. We then describe how to apply MixEncoder for different tasks such as classification task and ranking task.

3.1 Problem Statement

Given a sentence pair, models are required to generate either a prediction or a ranking score. The former is known as a linear-probe classification task (Nguyen et al., 2016). For classification task, the training set consists of paired samples, $\{q_i, p_i, y_i\}_{i=1}^N$, where $y_i$ is the label of the sentence pair, $N$ is the size of the dataset, and $q_i, p_i$ denotes the query and the candidate, respectively. For ranking task, the samples in the training set can be denoted as $\{q_i, p_i, C_i\}_{i=1}^N$, where $p_i$ is the positive candidate for the $q_i$ while $C_i$ is a set of negative candidates.

3.2 Candidate Pre-computation

We describe how MixEncoder pre-compute each existing candidate into several context embeddings offline. Let the token embeddings of one candidate be $T_i = [t_1, \cdots, t_d]$. We experiment with two strategies to obtain $k$ context embeddings from these token embeddings: (1) prepending $k$ special tokens $\{S_i\}_{i=1}^k$ to $T_i$ before feeding $T_i$ into an Transformer encoder (Vaswani et al., 2017; Devlin et al., 2019), and using the output at these special tokens as context embeddings ($S$-strategy); (2) maintaining $k$ context codes (Humeau et al., 2020) to extract global features from the last layer output of the encoder by attention mechanism ($C$-strategy). The default configuration is $S$-strategy as it provides slightly better performance.

Suppose there are $N$ candidates; we use $E_0 \in \mathbb{R}^{N \times k \times d}$ to denote the pre-computed context embeddings of these candidates, where $d$ indicates the embedding size.

3.3 Query Encoding

During the online inference stage, for a query with $N$ candidates, models have to measure the relevance of $N$ query-candidate pairs. A typical cross-encoder repeatedly concatenates the query with each candidate and encodes it $N$ times. It leads to prohibitive computation costs. One of the most effective ways to reduce the computation is to reduce the encoding times of the query.

In this section, we first depict the overview of the query encoder. Then, we introduce the core component of our MixEncoder: interaction layer. It performs a light-weight candidate-to-query cross-attention to estimate relevance scores in a single pass of the query encoding, no matter how many candidates the query has.

3.3.1 Overview of Encoder

Take an encoder that consists of five Transformer layers $L_1, L_2, \ldots, L_5$ as an example. When encoding the incoming query online, we replace the second and fifth Transformer layers $L_2, L_5$ with two interaction layers, denoted as $I_2^1, I_5^5$. Now the encoder can be depicted as $\{L_1, I_2^1, L_3, L_4, I_5^5\}$, shown in Figure 2(b). These layers are applied to the incoming query sequentially to produce contextualized representations of the query and the candidates.

Formally, each Transformer layer $L_i(\cdot)$ takes the query token representations $q_{i-1} \in \mathbb{R}^{m \times d}$ from the previous layer and produces a new representation matrix $q_i = L_i(q_{i-1})$, where $m$ denotes the query length and $q_i \in \mathbb{R}^{m \times d}$.

Each interaction layer $I_i^j(\cdot)$ takes the query token representations $q_{i-1}$ from the previous layer as input, along with the context embeddings $E_{j-1}$.
and a set of state vectors $H_{j-1} \in \mathbb{R}^{N \times d}$ from the previous interaction layer (or cache):

$$ q_i, E_j, H_j = I^j_i(q_{i-1}, E_{j-1}, H_{j-1}). $$  

The output $E, H$ of the last interaction layer are fed into a classifier to generate predictions for each query-candidate pair.

### 3.3.2 Interaction Layer

This section describes the details of how the interaction layer generates candidate and query representations.

**Candidate Representation.** Given $q_{i-1}$ and $E_{j-1}$, layer $I^j_i$ performs a self-attention over $q_{i-1}$, and a candidate-to-query cross-attention over $q_{i-1}, E_{j-1}$ simultaneously, as shown in Figure 2(b). Formally, the query self-attention is conducted as

$$ Q_{i-1}, K_{i-1}, V_{i-1} = \operatorname{LN}(q_{i-1}), $$

$$ q_i = \operatorname{FFN}(\operatorname{Att}(Q_{i-1}, K_{i-1}, V_{i-1})), $$

where we write $\operatorname{LN}(\cdot)$ for a linear transformation, $\operatorname{FFN}(\cdot)$ for a feed-forward network and $\operatorname{Att}(Q, K, V)$ for a self-attention operation (Vaswani et al., 2017). The cross-attention is formulated as

$$ Q'_{j-1}, K'_{j-1}, V'_{j-1} = \operatorname{LN}(E_{j-1}), $$

$$ E_j = \operatorname{FFN}(\operatorname{Att}(Q'_{j-1}, [K'_{j-1}; K_{i-1}], [V'_{j-1}; V_{i-1}])). $$

By simply concatenating $K_{i-1}, V_{i-1}$ generated from the query with $K'_{j-1}, V'_{j-1}$ generated from the candidates, the cross-attention operation dominated by $Q'_{j-1}$ aggregates the semantics for each query-candidate pair and produces new context embeddings $E_j \in \mathbb{R}^{N \times d}$.

As shown in Eq.(3) and (5), the interaction layer separates the query encoding and the cross-attention, thus the candidates embeddings are transparent to query. This design allows encoding the query only once regardless of the number of its candidates.

**Query Representation.** As shown in Eq.(5), the context embedding matrix $E$ contains the semantics from both the query and the candidates. It can be used to estimate the relevance score of candidate-query pairs as

$$ s = \operatorname{LN}((\operatorname{Avg}(E))), $$

where $s \in \mathbb{R}^N$.

Since $E$ may not be sufficient to represent semantics for each candidate-query pair, we choose to maintain a separate embedding $h$ to represent the query. Concretely, we conduct an attention operation at each interaction layer and obtain a unique query state for each candidate.

We first employ a pooling operation followed by a linear transformation on $E_{j-1}$ and obtain $Q^* \in \mathbb{R}^{N \times d}$. Then, the query semantics w.r.t. the candidates are extracted as

$$ H^* = \operatorname{FFN}(\operatorname{Att}(Q^*, K_{i-1}, V_{i-1})), $$

where $K_{i-1}, V_{i-1}$ are generated by Eq.(2).
Next, the gate proposed by (Cho et al., 2014) is utilized to fuse $H^*$ with the query states $H_{j-1}$:

$$H_j = \text{Gate}(H^*, H_{j-1}), \quad (8)$$

where $H_j \in \mathbb{R}^{N \times d}$. Each row of $H_j$ stands for the representation of the incoming query with respect to one candidate.

### 3.4 Classifier

Let $H$ and $E$ denote the query states and the candidate context embeddings generated by the last interaction layer, respectively. For the $i$-th candidate, the representation of the query is the $i$-th row of $H$, denoted as $h_i$. The representation of this candidate is the mean of the $i$-th row of context embeddings $E$, denoted as $e_i$.

#### Classification Task:

For a classification task such as NLI, we concatenate the embeddings $h_i$ and $e_i$ with the element-wise difference $|h_i - e_i|$ (Reimers and Gurevych, 2019) and feed them into a feed-forward network:

$$\text{logits} = \text{FFN}(h_i, e_i, |h_i - e_i|). \quad (9)$$

The network is trained to minimize a cross entropy loss.

#### Ranking Task:

For ranking tasks such as passage retrieval, we estimate the relevance score of candidate-query pairs as:

$$s_i = h_i \cdot e_i, \quad (10)$$

where $\cdot$ denotes dot product. The network is optimized by minimizing a cross-entropy loss in which the logits are $s_{i_1}, \cdots, s_{i_N}$.

### 3.5 Time Complexity

Table 1 presents the time complexity of the Dual-BERT, Cross-BERT, and our proposed MixEncoder. We can observe that the dual-encoder and MixEncoder support offline pre-computation to reduce the online time complexity. During the online inference, the query encoding cost term $(hq^2 + h^2q)$ of both Dual-BERT and MixEncoder does not increase with the number of candidates, since they conduct query encoding only once. Moreover, the MixEncoder’s query-candidate term $\sum_{c=0}^{N_c}(c + q + h)hc$ can be reduced by setting $c$ as a small value, which can further speed up the inference.

4 EXPERIMENTS

#### 4.1 Datasets

To fully evaluate the proposed MixEncoder, we conduct an empirical evaluation on four paired-input datasets, including natural language inference (NLI), information retrieval, and utterance selection for dialogue.

**MNLI** (Multi-Genre Natural Language Inference) (Williams et al., 2018) is a crowd-sourced classification dataset. It contains sentence pairs annotated with textual entailment information.

**MS MARCO Passage Reranking** (Nguyen et al., 2016) is a large collection of passages collected from Bing search logs. Given a query, the goal is to rank provided 1000 passages. We use a subset of training data as a small value, $c$. Following previous work, we evaluate the models on the 6980 development queries (Khattab and Zaharia, 2020; Gao et al., 2020).

**DSTC7** (Yoshino et al., 2019) is a chat log corpus contained in the DSTC7 challenge (Track 1). It consists of multi-turn conversations where one partner seeks technical support from the other.

**Ubuntu V2** (Lowe et al., 2015) is a popular corpus similar to DSTC7. It is proposed earlier and contains more data than DSTC7.

These four datasets have the same form that every sample in the dataset contains one text and several candidates. The statistics of these datasets are detailed in Table 2. We use accuracy to evaluate the classification performance on MNLI. For other datasets, MRR and recall are used as evaluation metrics.

#### 4.2 Baselines

MixEncoder is compared to following baselines: Cross-BERT refers to the original BERT (Devlin et al., 2019). We take the output at CLS token as the representation of the pair. This embedding is fed into a feedforward network to generate logits for either classification tasks or matching tasks. DualBERT (Sentence-BERT) is proposed by Reimers et al. (Reimers and Gurevych, 2019). This model uses siamese architecture and encodes text pairs separately. Deformer (Cao et al., 2020) is a decomposed Transformer, which utilizes lower layers to encode query and candidates separately and then uses upper layers to encode text pairs together. We followed the settings reported in the original
Table 1: Time Complexity of the attention module in MixEncoder, Dual-BERT and Cross-BERT. We use $q$, $d$ to denote query and candidate length, respectively. $h$ indicates the hidden layer dimension. $N_c$ indicates the number of candidates for each query and $k$ indicates the number of context embeddings for each candidate.

| Model       | Total ($N_c = 1$) | Pre-computation ($N_c = 1$) | Online                  |
|-------------|-------------------|-----------------------------|-------------------------|
| Dual-BERT   | $h(d^2 + q^2) + h^2(d + q)$ | $hd^2 + h^2d$ | $hd^2 + h^2q$ |
| Cross-BERT  | $h(d + q)^2 + h^2(d + q)$ | 0 | $N_c(h(q + d)^2 + h^2(q + d))$ |
| MixEncoder  | $h(d^2 + q(q + k) + k^2) + h^2(d + q + k)$ | $hd^2 + h^2d$ | $h(q + kN_c) + k^2N_c + h^2(q + kN_c)$ |

\[= \begin{align*}
&= h(q + kN_c) + k^2N_c + h^2(q + kN_c) \\
&= \text{Time for computations on the GPU and exclude time for text reprocessing and moving data to the GPU.}
\]

5 Results

Table 4 shows the experimental results of Dual-BERT, Cross-BERT, existing late interaction models and three variants of MixEncoder on four datasets. We measure the inference time of all the baseline models and present the results in Table 3.

5.1 Performance Comparison

Variants of MixEncoder. To study the effect of the number of interaction layers and that of the number of context embeddings per candidate, we present three variants in the tables, denoted as MixEncoder-a, -b and -c, respectively. Specifically, MixEncoder-a and -b set $k$ as 1. The former has interaction layer $I^{1}_{12}$ and the latter has layers $\{I^{10}_{1}, I^{11}_{1}, I^{12}_{2}\}$. MixEncoder-c has the same layers as MixEncoder-b but with $k = 2$.

Inference Speed. We conduct speed experiments to measure the online inference speed for all the baselines. Concretely, we samples 100 samples from MS MARCO. Each of samples has roughly 1000 candidates. We measure the time for computations on the GPU and exclude time for text reprocessing and moving data to the GPU.

Dual-BERT and Cross-BERT. The performance of the dual-BERT and cross-BERT are reported in

4.3 Training Details

While training models on MNLI, we follow the conventional practice that uses the labels provided in the dataset. While training models on other three datasets, we use in-batch negatives (Karpukhin et al., 2020; Qu et al., 2021) that considers the positive candidate of other queries in a training batch as negative candidates. For Cross-BERT and Deformer, that require exhaustive computation, we set batch size as 16 due to the limitation of computation resources. For other models, we set batch size as 64. All the models use one BERT (based, uncased) with 12 layers and fine-tune it for up to 50 epochs with a learning rate of 1e-5 and a linear scheduling. All experiments are conducted on a server with 4 Nvidia Tesla A100 GPUs which has 40 GB graphic memory.

Table 2: Statistics of experimental datasets.

| Dataset            | MNL1 | MS MACRO | DSTC7 | Ubuntu V2 |
|--------------------|------|----------|-------|-----------|
| **Train**          |      |          |       |           |
| # of queries       | 392,702 | 498,970 | 200,910 | 500,000   |
| Avg length of queries | 27   | 9        | 153   | 139       |
| Avg length of candidates | 14   | 76       | 20    | 31        |
| **Test**           |      |          |       |           |
| # of queries       | 9,796 | 6,898    | 1,000 | 50,000    |
| # of candidates per query | 1     | 1000     | 100   | 10        |
| Avg length of queries | 26   | 9        | 137   | 139       |
| Avg length of candidates | 14   | 74       | 20    | 31        |

Table 3: Time to evaluate 100 queries with 1k candidates. The Space used to cache the pre-computed embeddings for 1k candidates are shown.

| Model                | Time (ms) | Space (GB) |
|----------------------|-----------|------------|
|                      | 1k        | 1k         |
| Dual-BERT            | 7.2       | 0.3        |
| PolyEncoder-64       | 7.3       | 0.3        |
| PolyEncoder-360      | 7.5       | 0.3        |
| Deformer             | 27.0      | 8.6        |
| Cross-BERT           | 488.7     | 52.7       |
| MixEncoder-a         | 8.4       | 0.3        |
| MixEncoder-b         | 10.6      | 0.3        |
| MixEncoder-c         | 11.2      | 0.6        |
Table 4: Performance of Dual-BERT, Cross-BERT and three variants of MixEncoder on four datasets.

| Model          | MNLI Accuracy | Ubuntu R1@10 | Ubuntu MRR | DSTC7 R1@100 | DSTC7 MRR | MS MARCO R1@1000 | MS MARCO MRR(dev) | Speedup Times |
|----------------|---------------|--------------|------------|--------------|-----------|-------------------|-------------------|---------------|
| Cross-BERT     | 83.7          | 83.1         | 89.4       | 66.0         | 75.2      | 23.3              | 36.0              | 1.0x          |
| Dual-BERT      | 75.2          | 81.6         | 88.5       | 65.8         | 73.7      | 20.3              | 32.2              | 132x          |
| PolyEncoder-64 | 76.8          | 82.3         | 89.9       | 67.5         | 75.2      | 20.3              | 32.3              | 130x          |
| PolyEncoder-360| 77.3          | 81.8         | 88.6       | 65.7         | 73.4      | 20.5              | 32.4              | 127x          |
| CoBERT         | ×             | 82.8         | 89.3       | 67.2         | 74.8      | 22.8              | 35.4              | 35.2x         |
| Deformer       | 82.0          | 83.2         | 89.5       | 66.3         | 75.3      | 23.0              | 35.7              | 1.9x          |
| MixEncoder-a   | 77.5          | 83.2         | 89.5       | 67.3         | 74.7      | 19.8              | 31.6              | 113x          |
| MixEncoder-b   | 77.8          | 83.2         | 89.5       | 68.7         | 76.1      | 20.7              | 32.5              | 89.6x         |
| MixEncoder-c   | 78.4          | 83.3         | 89.5       | 66.8         | 74.9      | 19.3              | 31.0              | 84.8x         |

Table 5: Ablation analysis for MixEncoder-a and MixEncoder-b. (MRR)

| Variants | Ubuntu | DTSC7 |
|----------|--------|-------|
| Original | 89.5   | 74.7  |
| w/o H    | 88.9   | 74.0  |
| w/o E    | 89.2   | 74.8  |
| Eq. 6    | 89.1   | 72.3  |

the first two rows of Table 4. We can observe that the MixEncoder consistently outperforms the Dual-BERT. The variants with more interaction layers or more context embeddings generally yield more improvement. For example, on DSTC7, MixEncoder-a and MixEncoder-b achieves an improvement by 1.0% (absolute) and 2.4% over the Dual-BERT, respectively. Moreover, MixEncoder-a provides comparable performance to the Cross-BERT on both Ubuntu and DSTC7. MixEncoder-b can even outperform the Cross-BERT on DSTC7 (+0.9), since MixEncoder can benefit from a large batch size (Humeau et al., 2020). On MNLI, MixEncoder-a retains 92.6% of effectiveness of the Cross-BERT and MixEncoder-c can retain 93.7% of that. However, the effectiveness of the MixEncoder on MS MARCO is slight.

We can find that the difference of the inference time for processing samples with 1k candidates between the Dual-BERT and MixEncoder is minimal. Cross-BERT is 2 orders of magnitude slower than these models.

Late Interaction Models. From tables 3, 4, we can have following observations. First, among all the late interaction models, Deformer that adopts a stack of Transformer layers as the late interaction component consistently shows the best performance on all the datasets. This demonstrates the effectiveness of cross-attention in transformer layers. In exchange, Deformer shows limited speedup (1.9x) compared to Cross-BERT. Compared to the CoBERT and Poly-Encoder, our MixEncoder outperforms them on the datasets except MS MARCO. Although CoBERT consumes more computation than MixEncoder, it shows worse performance than MixEncoder on DSTC7 and Ubuntu. This demonstrates the effectiveness of the light-weight cross-attention, which can achieve a trade-off between the efficiency and effectiveness. However, on MS MARCO, our MixEncoder and poly-encoder lag behind the CoBERT with a large margin. We conjecture that our MixEncoder falls short of handling token-level matching. We will elaborate it in section 5.5.

5.2 Effectiveness of Interaction Layer Representations. We conduct ablation studies to quantify the impact of two key components (E and H) utilized in MixEncoder. The results are shown in Table 5. Every component results in a gain in performance compared to Dual-BERT. It demonstrates that our simplified cross-attention can produce effective representations for both the candidate and query. An interesting observation is that removing E can lead to a slight improvement on DSTC7. Moreover, we also implement MixEncoder based on Eq. 6 that a linear transformation is applied to E to estimate relevance scores, which leads to a drop in performance.

Varying the Interaction Layers. To verify the impact of the interaction layer, we perform ablation studies by varying the number and the position of layers. First, we use two interaction layers \( \{I_1^1, I_2^1\} \), and choose \( i \) from the set \( \{1, 2, 4, 6, 8, 10, 11\} \). The results are shown in Figure 3(b). We can find that MixEncoder on Ubuntu.
is insensitive to \( i \) while MixEncoder on DSTC7 can be enhanced with \( i = 11 \). Moreover, Figure 3(c) shows the results when MixEncoder has interaction layers \( \{ I^1_1, I^2_{i+1}, \ldots, I^3_{12-i} \} \). Increasing interaction layers cannot always improve the ranking quality. On Ubuntu, replacing all the transformer layers provides close performance to that with the last layer replaced. On DSTC7, the performance of MixEncoder achieves a peak with last three layer replaced by our interaction layers.

5.3 Candidate Pre-computation

We study the effect of the number of candidate embeddings, denoted as \( k \), and the pre-computation strategies introduced in section 3.2. Specifically, We choose the value of \( k \) from the set \( \{1, 2, 3, 10\} \) with one interaction layer \( I^1_1 \). From Figure 3(c), we can observe that as \( k \) gets larger, the performance of the MixEncoder increases first, and then descends. Moreover, two pre-computation strategies have different impacts on the model performance. S-strategy generally outperforms C-strategy with the same \( k \).

5.4 In-batch Negative Training

We change the batch size and show the results in Figure 3(d). It can be observed that increasing batch size contributes to good performance. Moreover, we have the observation that models may fail to diverge with a small batch size. Due to the limitation of computation resources, we set the batch size as 64 for our training.

5.5 Error Analysis

In this section, we take a sample from MS MARCO to analyze our errors. We observe that MixEncoder falls short of detecting token overlapping. Given the query "foods and supplements to lower blood sugar", MixEncoder fails to pay attention to the keyword "supplements" which appears in both the query and the positive candidate. We conjecture that this drawback is due to the pre-computation that represents each candidate into \( k \) context embeddings. It lose the token-level feature of the candidates. On the contrary, ColBERT caches all the token embeddings of the candidates and estimate relevance scores based on token-level similarity.

6 Conclusion

In this paper, we propose MixEncoder which provides a good trade-off between the performance and efficiency. MixEncoder involves a light-weight cross-attention mechanism which allows us to encode the query once and process all the candidates in parallel. We evaluate MixEncoder with four datasets. The results demonstrate that MixEncoder can speed up sentence pairing by over 113x while achieving comparable performance as the more expensive cross-attention models.
7 Limitations

Although MixEncoder is demonstrated effective, we recognize that MixEncoder does not perform well on MS MARCO. It indicates that our MixEncoder falls short of detecting token overlapping, since it may lose token level semantics of candidates during pre-computation. Moreover, the MixEncoder is not evaluated on a large scale evaluation dataset, such as an end-to-end retrieval task, which requires model to retrieve top-\(k\) candidates from millions of candidates (Qu et al., 2021; Khattab and Zaharia, 2020).

References

Qingqing Cao, Harsh Trivedi, Aruna Basalasubramanian, and Niranjan Balasubramanian. 2020. DeFormer: Decomposing pre-trained transformers for faster question answering. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 4487–4497, Online. Association for Computational Linguistics.

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, Online. Association for Computational Linguistics.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, pages 103–111, Doha, Qatar. Association for Computational Linguistics.

Alexis Conneau and Douwe Kiela. 2018. Senteval: An evaluation toolkit for universal sentence representations. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018, European Language Resources Association (ELRA).

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.

Luyu Gao, Zhuyun Dai, and Jamie Callan. 2020. Modularized transformer-based ranking framework. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 4180–4190, Online. Association for Computational Linguistics.

Jiafeng Guo, Yinqiong Cai, Yixing Fan, Fei Sun, Ruqing Zhang, and Xueqi Cheng. 2022. Semantic models for the first-stage retrieval: A comprehensive review. ACM Transactions on Information Systems (TOIS), 40(4):1–42.

Zhe Hu, Zuohui Fu, Yu Yin, and Gerard de Melo. 2021. Context-aware interaction network for question matching. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 3846–3853, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In 22nd ACM International Conference on Information and Knowledge Management, CIKM’13, San Francisco, CA, USA, October 27 - November 1, 2013, pages 2333–2338. ACM.

Samuel Humeau, Kurt Shuster, Marie-Anne Lachaux, and Jason Weston. 2020. Poly-encoders: Architectures and pre-training strategies for fast and accurate multi-sentence scoring. In 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net.

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, Online. Association for Computational Linguistics.

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, SIGIR 2020, Virtual Event, China, July 25-30, 2020, pages 39–48. ACM.

Jian Li, Jieming Zhu, Qiwei Bi, Guohao Cai, Lifeng Shang, Zhenhua Dong, Xin Jiang, and Qun Liu. 2022. MINER: Multi-interest matching network for news recommendation. In Findings of the Association for Computational Linguistics: ACL 2022, pages 343–352, Dublin, Ireland. Association for Computational Linguistics.

Ryan Lowe, Nissan Pow, Iulian Serban, and Joelle Pineau. 2015. The ubuntu dialogue corpus: A large dataset for research in unstructured multi-turn dialogue systems. In Proceedings of the SIGDIAL 2015 Conference, The 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue, 2-4 September 2015, Prague, Czech Republic, pages
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.

Tri Nguyen, Mir Rosenberg, Xia Song, Jianfeng Gao, Saurabh Tiwary, Rangan Majumder, and Li Deng. 2016. MS MARCO: A human generated machine reading comprehension dataset. In Proceedings of the Workshop on Cognitive Computation: Integrating neural and symbolic approaches 2016 co-located with the 30th Annual Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain, December 9, 2016, volume 1773 of CEUR Workshop Proceedings. CEUR-WS.org.

Ping Nie, Yuyu Zhang, Xiubo Geng, Arun Ramanurthy, Le Song, and Daxin Jiang. 2020. DC-BERT: decoupling question and document for efficient contextual encoding. In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020, pages 49–58. ACM.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Koichiro Yoshino, Chiori Hori, Julien Perez, Luis Fernando D’Haro, Lazaros Polymenakos, R. Chulaka Gunasekara, Walter S. Lasecki, Jonathan K. Kummerfeld, Michel Galley, Chris Brockett, Jianfeng Gao, Bill Dolan, Xiang Gao, Huda AlAmri, Tim K. Marks, Devi Parikh, and Dhruv Batra. 2019. Dialog system technology challenge 7. CoRR, abs/1901.03461.

Linhao Zhang, Dehong Ma, Sujian Li, and Houfeng Wang. 2021. Do it once: An embarrassingly simple joint matching approach to response selection. In Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 565–575, Online. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. 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 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Ruiyang Ren, Yingqi Qu, Jing Liu, Wayne Xin Zhao, QiaoQiao She, Hua Wu, Haifeng Wang, and Ji-Rong Wen. 2021. RocketQAv2: A joint training method for dense passage retrieval and passage re-ranking. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 2825–2835, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA, pages 5998–6008.

A Appendix

This is an appendix.