DiPair: Fast and Accurate Distillation for Trillion-Scale Text Matching and Pair Modeling

Jiecao Chen[*], **Liu Yang, Karthik Raman, Michael Bendersky**
Jung-Jung Yeh, Yun Zhou, Marc Najork, Danyang Cai, Ehsan Emadzadeh
Google Research

Abstract

Pre-trained models like BERT (Devlin et al., 2018) have dominated NLP / IR applications such as single sentence classification, text pair classification, and question answering. However, deploying these models in real systems is highly non-trivial due to their exorbitant computational costs. A common remedy to this is knowledge distillation (Hinton et al., 2015), leading to faster inference. However – as we show here – existing works are not optimized for dealing with pairs (or tuples) of texts. Consequently, they are either not scalable or demonstrate subpar performance. In this work, we propose **DiPair** — a novel framework for distilling fast and accurate models on text pair tasks. Coupled with an end-to-end training strategy, DiPair is both highly scalable and offers improved quality-speed tradeoffs. Empirical studies conducted on both academic and real-world e-commerce benchmarks demonstrate the efficacy of the proposed approach with speedups of over 350x and minimal quality drop relative to the cross-attention teacher BERT model.

1 Introduction

Modeling the relationship between textual objects is critical to numerous NLP and information retrieval (IR) applications (Li and Xu, 2014). This subsumes a number of different problems such as textual entailment, semantic text matching, paraphrase identification, plagiarism detection, and relevance modeling. For example, modeling the relationship between queries and documents / ad keywords is central to search engines / digital advertisement systems (Li and Xu, 2014; Guo et al., 2019).

Recently neural network-based models have demonstrated large gains in this space (Hu et al., 2014; Pang et al., 2016). In particular, the Transformer / BERT family of models (Devlin et al., 2018; Lan et al., 2019; Liu et al., 2019; Clark et al., 2020) have set a new bar for these semantic text matching problems. However, the computational costs of these models have proven to be prohibitively expensive, thus limiting their use in real-world applications (Frankle and Carbin, 2019). For example, on the e-commerce relevance-scoring task (P2T-REL dataset) discussed in Sec. 4.1, scoring the (trillion+) text pairs would take years.

One popular remedy is to distill these expensive *teacher* models (Hinton et al., 2015) into lightweight *student* models. Training these *students* using examples labeled by the *teacher* has been shown to maintain quality while enabling faster inference. The key to the effectiveness of distillation techniques is a good trade-off between student quality and inference speed.

However, as we show here, existing knowledge distillation techniques (Sanh et al., 2019; Jiao et al., 2019; Turc et al., 2019; Tang et al., 2019) fall short on the quality-speed trade-off when dealing with pairs of texts. On one hand, approaches that model the texts jointly (*i.e.*, using cross-attention) even one as highly optimized as BERT-TINY (Turc et al., 2019) are still orders of magnitude too slow.

On the other hand, techniques that model the texts independently such as the *dual-encoder* models[1] (Das et al., 2016; Johnson et al.; Chidambaram et al., 2019; Cer et al., 2018; Henderson et al., 2017; Reimers and Gurevych, 2019) are able to run efficient inference on large-scale text pairs. By exploiting the independence of the texts, these techniques can significantly speed up inference by caching/indexing embeddings of individual texts. However, this speedup comes at a significant cost – with sharply reduced scoring quality.

The key drawback here is that these independent models lack the ability to mimic the cross-attention

[*]Correspondence to chenjiecao@google.com

[1]These models encode the two texts separately and then combine them via a lightweight dot product / cosine.

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 2925–2937

enabled teachers and model the joint nuances and facets of the texts. As a motivating example, consider the ecommerce term relevance-scoring task. For the product "Black Sport Nike Shoes for Boys Size Wide", terms such as "black", "wide footwear" and "nike shoes" are all relevant. However, enforcing similarity between the independently modeled term and product will lead to the embeddings of "black" and "nike shoes" being incorrectly considered similar.

Motivated by this, we propose **DiPair** for fast and accurate distillation of large-scale text matching and pair modeling. DiPair aims to combine the best of both worlds: Like dual-encoder models, it leverages common pre-computation, while at the same time modeling the text jointly – with cross-attention – using multiple contextual embeddings for each text. In particular, we extract a small fraction of the output token embeddings from each text, and then jointly model this smaller "sequence" using a transformer *head* (we use the term *head* to refer to the component that consumes the outputs of a dual-encoder model, see Figure 2). We demonstrate that a two-stage, end-to-end training allows the proposed DiPair model to learn richer multifaceted semantic representations of the text pairs. The resulting DiPair model is 350x+ faster with minimal quality drop relative to the teacher on academic and real-world e-commerce datasets.

In summary, our main contributions include:

- **DiPair**: A new framework for distilling fast, accurate models on text pair tasks. Its advantages include: 1) Generic framework applicable across numerous applications involving pairwise/n-ary textual input. To the best of our knowledge, this is among the first few works tackling this problem. 2) Highly practical solution with limited storage and computation needs that scales to trillions of examples. 3) Large speedups for model inference – 350x+ faster relative to the BERT-base teacher and 8x faster than previous highly optimized benchmarks (Turc et al., 2019).
- A two-stage, end-to-end training scheme enables an improved quality-speed tradeoff as shown in Fig. 1.
- Evidence that (self and cross) attention is important for student models when it comes to distilling from teachers like BERT.
- Extensive experiments on academic and real-world e-commerce datasets demonstrate that DiPair can lead to fast and accurate models that outperform existing techniques on text matching and pair modeling.

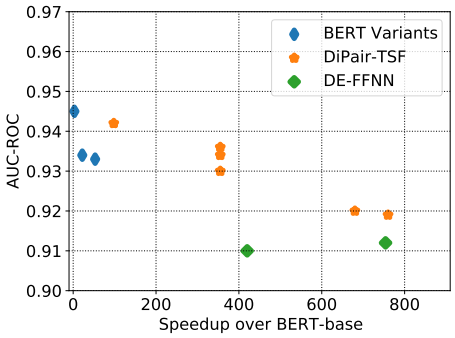


Figure 1: Inference speed vs quality trade-off for three different representative approaches (see Sec. 4.2 for a more detailed description): DE-FFNN refers to dual-encoder + a feedforward neural network head; BERT variants refers to different cross-attention BERT-based student models released by Turc et al. (2019); DiPair-TSF refers to the DiPair model with a transformer head. The metrics and inference speedup are evaluated with the Q2P-MAT dataset (see Sec. 4.1). Note that the DiPair and DE-FFNN are varied using only the head in this plot for a fair comparison.

2 Related Work

Text Pair Modeling and Matching. A large variety of neural models have been proposed for text pair tasks such as matching and similarity scoring (Huang et al., 2013; Hu et al., 2014; Pang et al., 2016; Guo et al., 2016; Yang et al., 2016; Mitra et al., 2017; Xiong et al., 2017; Rao et al., 2019). These models can be broadly classified into *representation-focused* models (or *dual-encoder* models) (Huang et al., 2013; Hu et al., 2014) and *interaction-focused* models (Pang et al., 2016; Guo et al., 2016; Yang et al., 2016; Mitra et al., 2017; Xiong et al., 2017), where the former involves encoding the individual text separately while the latter models the pair jointly (often involving some interaction / attention model). In recent years, Transformer (Vaswani et al., 2017) based models like BERT (Devlin et al., 2018) leveraged cross-attention to achieve impressive performance gains on several text pairs tasks including natural language inference (Bowman et al., 2015), sentence pair classification and relevance scoring. As shown in several previous research (Pang et al., 2016; Guo et al., 2016; Yang et al., 2016; Mitra et al., 2017; Xiong et al., 2017; Devlin et al., 2018), interaction-focused models usually achieve better

performances for text pair tasks. However, it is difficult to serve these types of models for applications involving large inference sets in practice. On the other hand, text embeddings from dual encoder models can be learned independently and thus pre-computed, leading to faster inference efficiency but at the cost of reduced quality. Early work like (Wang and Jiang, 2017) uses attention to aggregate the two sequences of word embeddings, and a CNN model is then applied to extract the final representation. This method is relatively expensive as it requires to store the whole sequences of word embeddings and a full cross-attention operation has to be performed. Recently the PreTTR model (MacAvaney et al., 2020) aimed to reduce the query-time latency of deep transformer networks by pre-computing part of the document term representations. However, their model still required modeling the full document/query input length in the head, thus limiting inference speedup. Another recent work is Poly-encoders (Humeau et al., 2020) which shared some similar motivations. However, Poly-Encoders makes strong assumptions on the input data property thus limiting its applicability (Appendix C demonstrates this quality drop on a standard text matching task).

Knowledge Distillation. Our research is an example of knowledge distillation in neural networks (Hinton et al., 2015; Sun et al., 2019; Sanh et al., 2019). The idea of knowledge distillation is to transfer information from a heavily-parameterized and accurate teacher model to a lightweight student model for faster inference. Tang et al. (2019) proposed to distill knowledge from BERT to a single-layer BiLSTM model. TinyBERT (Jiao et al., 2019) performs knowledge distillation into transformers in two-stage learning including pre-training and task-specific fine-tuning. Turc et al. (2019) proposed Pre-trained Distillation, which shows task-specific distillation on an unlabeled transfer set is helpful to improve the student model performance. Key differences between our work and these approaches are that we focus on model distillation for text pair inputs and speeding up inference while aiming to match the teacher's performances.

Model Quantization and Parameter Pruning. Another line of research loosely connected to our work is to reduce inference time via pruning less significant weights and/or converting the model to low-precision (aka quantization) (Han et al., 2016; Howard et al., 2017; Iandola et al., 2016; Renda et al., 2020; Frankle and Carbin, 2019). Effective in many applications, those approaches, however, often only lead to less than 20x speedup and therefore do not scale to many tasks with pairwise input.

3 Our Approach

3.1 Method Overview

Figure 2 provides an overview of the proposed DiPair model. First, a transformer-based *dual-encoder* model is applied to the input pair; the output of an encoder is a sequence of token embeddings, which has the same sequence length as the tokenized input text. We then *truncate* the output sequences by only taking the first N and M token embeddings from the left and right inputs, respectively; the next step is to project those selected token embeddings into lower dimensions and merge them to form the new input sequence. The merged input sequence is then fed into the transformer (or an FFNN) *head*, and the first token embedding of the output sequence of the head is used as the representation of the initial input pair.

Note that, the dual-encoder will process the full-length input sequences. At the same time, the head only consumes a sequence of length $(N + M)$, which is typically much smaller than the length of the input sequences and ensures efficient execution of the head.

To create the training data for our proposed model, we use an expensive teacher model (e.g., a 12-layer BERT fine-tuned with human-rated data) to annotate a set of unlabeled text pairs (a.k.a. distillation set). The dual-encoder part of our model is initialized from the first few layers of a pre-trained BERT, and a novel *two stage training strategy* (see Sec. 3.7) is applied to boost the performance further. We defer more details of data specific model distillation to Sec. 4.3.

We now discuss each component of the proposed architecture in detail.

3.2 Dual-Encoder

A dual-encoder is the key component of our proposed architecture, and we initialize our dual-encoder from pre-trained BERT (or tinyBERT, ALBERT, etc.). Our basic assumption is that the number of pairs is much larger than the set of unique inputs to the left or right encoders, and the bottleneck of serving our model is to run inference on the pairs with the head. Our proposed architecture, therefore, has an important benefit: increasing the

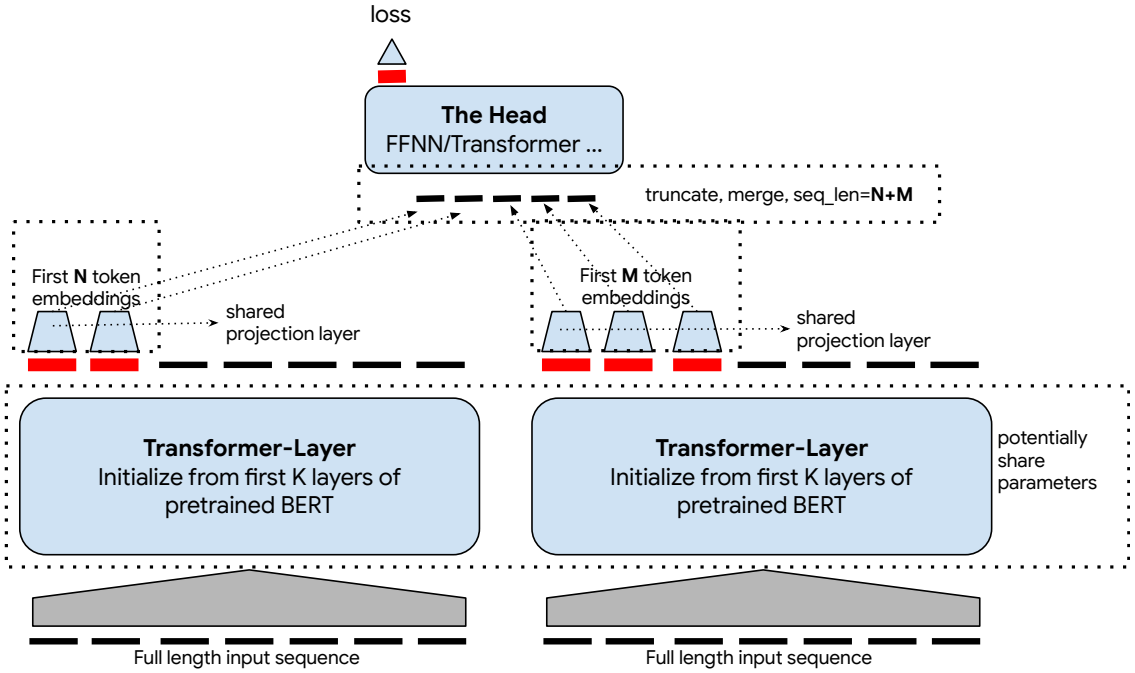


Figure 2: The architecture of the DiPair model.

model capacity does not increase the inference time as we can keep the head the same but use more expensive encoders. Figure 3b shows that increasing the number of layers of the encoders will often lead to better model performance.

3.3 Truncated Output Sequences

This is the key step to speed up the model serving. Recall that the running time of a transformer-based model quadratically depends on the input sequence length. One of the most effective ways to reduce the running time is to reduce the input sequence length. However, as Table 4 reveals, blindly truncating the input to a BERT model will lead to a quick performance drop. Our key intuition is that, by using a dual-encoder + head architecture, we can focus on reducing the inference time of the head, instead of speeding up the encoders.

Therefore, we still use the full-length input sequences in our encoders, but aggressively reduce the input sequence length to the head. To be more concrete, before merging the outputted sequences from the two encoders, we take the first N and M token embeddings from the left and the right sequences, respectively; This truncation technique has several benefits:

- It significantly speeds up the inference with the head, as the time complexity of transformer layers is quadratic w.r.t. the input sequence length.
- It significantly reduces the amount of data we need to cache. Only the first few token embeddings need be stored as the output of the encoders.
- N and M can be tuned to reflect the desired effectiveness and efficiency trade-off for a particular problem domain.

It is important to note that due to the end-to-end architecture of our model, even though we only use $(N + M)$ token embeddings from the output of the dual-encoder, the model learns to push the information of the input text to the first $(N + M)$ embeddings (thanks to the transformer layers, those selected token embeddings can interact with other token embeddings, and can be viewed as a summary of the full-length input sequences).

3.4 Projection Layer

For each encoder, we add a projection layer to project each token embedding to a lower dimension. A projection layer is shared within an encoder, but different encoders may use different projection layers. There are two purposes of adding the projection layers:

- Reduce storage. To run the inference with the proposed architecture, we need to cache all

the outputs from the encoders.

- Speed up the inference with the head. The time complexity of a transformer linearly depends on the embedding dimension.

In Table 5, we show that by choosing a proper projection layer, we can significantly reduce the embedding dimension with almost no quality drops.

3.5 Transformer-Based Head

After the projection layer, we merge the $N + M$ projected token embeddings into one sequence and feed it into the head. Like the BERT model, we also add position embeddings and segment embeddings to help the transformer head better aggregate the input sequence. The first token embedding (i.e., CLS embedding) of the transformer head is used as the final representation of the input pair.

Another advantage of using a head is that the head is *tokenization-free*: the input to the head is purely float tensors, and we do not need to preprocess/ tokenize the input text. This may lead to an additional speedup.

It is worth mentioning that a feedforward neural network (FFNN) can also be used as a head. An FFNN is faster than a transformer-head and often gives reasonable performance (though worse than a transformer head). See the experimental section (Sec. 4.6) for more discussion on these trade-offs.

3.6 Task Specific Losses

In the standard dual-encoder model and the recent Poly-Encoders (Humeau et al., 2020) work, the dot product between the embeddings is a scalar, which is not suited for tasks beyond regression/binary classification. On the other hand, our proposed architecture outputs a representation of the input pair and is therefore compatible with a wide range of loss functions.

3.7 A Two-Stage Training Approach

It turns out that directly training the proposed models often leads to sub-optimal results (see Sec. 4.7 for more evidence). This is primarily because adding non-trivial layers on top of a well pre-trained dual-encoder during training may corrupt the knowledge that has been preserved in the dual-encoder. To address this issue, we propose to use a *two-stage training strategy*: we first freeze the dual-encoder part and only train the newly added parameters until convergence; we then unfreeze the dual-encoder and further train the entire model. A similar training strategy can be found in e.g., (Wang et al., 2019).

3.8 Extension to n-Ary Tuple

Unlike the models proposed in the recent works (MacAvaney et al., 2020; Humeau et al., 2020) where only pairs can be supported, our proposed architecture trivially extends to the scenario where we have n-ary tuple of textual objects $(a_1, a_2, \ldots, a_n)$ as the model input, as we can simply replace the dual-encoder model with an n-encoder model. This feature is useful in many applications, such as QA tasks with context, query to document scoring tasks with personalized information.

4 Experiments

In this section, we conduct experimental studies. We aim to answer the following questions through our experiments:

- RQ1: How well does our proposed architecture perform compared with other strong baseline approaches? Compared with the teacher, how much faster are our methods (Sec. 4.5)?
- RQ2: Compared with FFNN heads, is the transformer head essential to reduce the distillation gap (Sec.4.6)?
- RQ3: How does two-stage training affect the final model performance (Sec. 4.7)?
- RQ4: How would the proposed dual-encoder+head architecture be affected by other hyper-parameters of different components (Sec. 4.8).

4.1 Datasets

We evaluate our proposed methods on two datasets (Table 1 provides an overview):

- Q2P-MAT is a binary classification task derived from the MSMARCO Passage Ranking data[2]. Given a *(query, passage)* pair, the goal is to predict whether the passage contains the answer for the query. We measure the model performance using AUC-ROC. Appendix B lists more details.
- P2T-REL is a regression task on a real-world ecommerce dataset. Given a *(product, term)* pair, the goal is to predict the relevance of the term to the product. We measure the model performance using Pearson correlation with

[2] https://microsoft.github.io/MSMARCO-Passage-Ranking/

the human judgments. Title and description are used as the product features. Appendix A provides several examples of *(product, term)* pairs.

4.2 Baseline Approaches

There exist many knowledge distillation (see Sec. 2 for more details) works, but none of them has been optimized for pairwise input. We choose to compare our DiPair approach with the fastest BERT-based student model (Turc et al., 2019) we are aware of, and our model is at least **8x** faster (see Table 3b). We also compare our proposed approach with several other strong baselines:

- BERT-TINY: the fastest version of BERT released in (Turc et al., 2019). This model has 2 layers with 128D word embeddings and 2-head transformer. It is claimed to be 52x faster than BERT-base (on TPU), and to the best of our knowledge, this is faster than any other BERT-based student models in the literature.
- DE-COS: BERT-based Dual-Encoder model. Cosine between left/right CLS embeddings is used as the similarity score.
- DE-FFNN: BERT-based Dual-Encoder model. FFNN (Feedforward Neural Networks) is used to aggregate the left/right CLS embeddings into a similarity score. Unless otherwise stated, we fix the FFNN to be 2-Layer with dimensions x128x128. The input to the FFNN has dimension $768 + 768 = 1536$.
- DIPAIRTSF: our proposed model, BERT-based Dual-Encoder, with a transformer-based head. N and M refer to the output sequence lengths (see Figure 2). In all experiments, we fix our head to be 2-Layer, 1-Head, 1024D intermediate size. The value of hidden_size (i.e., the dimension of the input token embeddings) is decided by the output of the projection layer.
- DIPAIRFFNN: this is similar to DIPAIRTSF; the only difference is that the transformer-based head is replaced with an FFNN. The input to the FFNN has dimension $(N + M) *$ hidden_size (N, M defined in Figure 2). We use 2-Layer FFNN with dimensions x128x128 unless otherwise stated.

In all the aforementioned models (except BERT-TINY), the dual-encoder is initialized from the first K layers of the pre-trained BERT model as well as the token embedding matrix. Unless otherwise stated, we fix K=1 for P2T-REL and K=4 for Q2P-MAT. The Left encoder and the right encoder will share parameters. For models with a projection layer, we use D to represent the dimension of the projected result.

4.3 Model Distillation

Teacher Models For Q2P-MAT, we use Google's public 12-layer BERT-base pre-trained model, and fine-tune it with the 1.1M labeled query to passage pairs.

On the other hand, for P2T-REL data, we pre-train a 12-layer BERT-based model with a customized vocabulary of size 80K, using user interaction data. We use the default parameters released in the public BERT code.[3] We then fine-tune the pre-trained model using the 393K product to term pairs.

For both teachers, we use the following cross-entropy loss,

$$-\sum_i (y_i \log p_i + (1 - y_i) \log(1 - p_i)) \quad (1)$$

where y_i is the label and p_i is computed via applying a sigmoid function on the teacher's logits z_i. This loss function works for both regression problems and binary classification problems.

Distillation Inspired by Hinton et al. (2015), we use sigmoid(z_i/T) to create soft labels to annotate the distillation sets, where z_i is teacher's logits and T is known as the temperature. In our experiment, we fix $T = 1$. We then apply the cross entropy loss as detailed in Equation (1).

4.4 Experimental Setup

Our code is implemented with TensorFlow [4] and we use TPUv3 in all of our experiments. We use AdamW optimizer following the public BERT code. The warmup step is fixed to be 50k. Other parameters of the optimizer are identical to the default values set in the public BERT code (weight_decay_rate=0.01, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e^{-6}$).

We tune some other key hyper-parameters using the validation sets. We try multiple (learning rate, batch size) combinations and choose the best ones. In the two-stage training, the models are less

[3] Available in https://github.com/google-research/bert.

[4] https://www.tensorflow.org/

Data	P2T-REL				Q2P-MAT			
Item	Distill	Train	Valid	Test	Distill	Train	Valid	Test
# of pairs	300M	393K	12.8K	12.8K	40M	1.1M	12.8K	12.8K
AvgLen product/passage	107.6	84.3	83.5	82.1	55.5	56.0	53.6	53.8
AvgLen term/query	1.49	1.32	1.32	1.31	6.37	6.03	6.00	6.03

Table 1: Datasets statistics.

sensitive to learning rates in the first stage, and we set the learning rate as 5e-5; we then train the models until they converge. In the second stage of training, the learning rate is set to be 5e-5 in DIPAIRTSF, DE-COS, DIPAIRFFNN; we use batch size 512 and 4x4 TPU topology. For BERT-TINY, we use batch size 128, learning rate 2e-6, and 2x2 TPU topology. All other hyperparameters related to model architecture are specified in Sec. 4.2.

4.5 Main Results

Table 2 and Table 3 present the experimental results on P2T-REL and Q2P-MAT datasets, respectively. Among all the student models with dual-encoder architecture, DIPAIRTSF consistently achieves the best performance. For the Q2P-MAT dataset, DIPAIRTSF achieves similar AUC_ROC to BERT-TINY; however, it achieves a **8x** speedup.

Among all the student models, DE-COS is the fastest one as it only requires dot product during inference. However, it has the worst performance, indicating that using Cosine function alone does not allow enough interaction between the input sequences embeddings.

4.6 Effectiveness of Transformer Head

To verify the importance of using a transformer-based head, we vary #params in the heads of DIPAIRTSF, DE-FFNN and DIPAIRFFNN. Table 4 presents the experimental results.

Comparing rows 1 and 2 in Table 4, the model quality of DIPAIRTSF can be improved by increasing the head input sequences lengths (N and M), although at the cost of longer inference time. On the other hand, rows 3-5 show that increasing #Params in FFNN head (e.g., using larger dimensions, more layers) does not lead to significant quality improvement for DIPAIRFFNN; even when the #Params of the FFNN head is 4x more than the transformer head, the model quality of DIPAIRTSF is still considerably superior to that of DIPAIRFFNN(cf. rows 2 and 5). A similar conclusion can be made for DE-FFNN (rows 6-8).

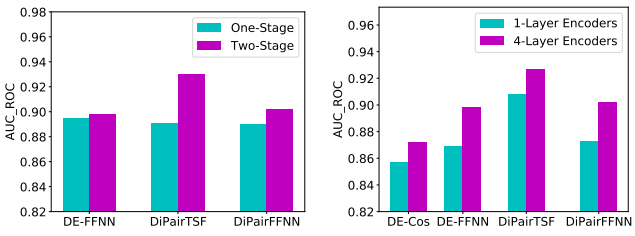
Another interesting observation is that even with more input information and more parameters, DIPAIRFFNN does not generate higher AUC_ROC than DE-FFNN. This might suggest that FFNN is not powerful enough to aggregate the input information effectively.

Overall, Table 4 illustrates the importance of using a transformer head if we want to achieve high model quality: Unlike FFNN-based heads, where we could not further improve the model via increasing #Params, a transformer-based head has more headroom to reduce the distillation gap further, and the desired quality-speed trade-off can be easily achieved by adjusting the values of N and M.

4.7 Effect of Two-Stage Training

Figure 3a shows that two-stage training, which is discussed in Section 3.7 has positive effects on all the methods we test. When the head is transformer-based, the two-stage training plays an important role: the AUC_ROC improves from 0.891 to 0.930.

On the other hand, the gain introduced by using two-stage training is less significant in other approaches such as DE-FFNN and DIPAIRFFNN. This might be because FFNN is generally easier to train than transformer-based models, and thus initialization choices play a lesser role.



(a) (b)

Figure 3: (a) One-stage training v.s. two-stage training. Projection dimension D=256. (b) Different # of layers in encoders. D=256

4.8 Model Ablation Studies

Varying the Encoder Layers Figure 3b shows that we can improve the model performance by increasing the number of layers in the encoders. Since the heads remain the same, and the number of pairs is often far greater than the number of the unique items needed to be encoded, the total inference time will not increase accordingly.

Model Settings	Pearson (valid)	Delta (valid)	Pearson (test)	Delta (test)	Speedup
Teacher (BERT-base)	0.757	-0%	0.757	-0%	1x
DE-FFNN	0.682	-9.9%	0.677	-11.6%	3129x
DE-COS	0.678	-10.4%	0.669	-11.6%	**3990x**
DIPAIRFFNN	0.696	-8.1%	0.697	-7.9%	2128x
DIPAIRTSF	**0.732***	**-3.3%***	**0.731***	**-3.4%***	362x

(a) Compare with Dual-Encoder based model.

Model Settings	Pearson (valid)	Delta (valid)	Pearson (test)	Delta (test)	Speedup
BERT-TINY	0.644	-12.0%	0.640	-11.3%	53x
DIPAIRTSF	**0.732***	**-3.3%***	**0.731***	**-3.4%***	**362x**

(b) Compare with BERT-based student model.

Table 2: Main results for P2T-REL data. Entries marked with * are significant (p-value < 0.05, w.r.t. the closest baseline, following (Berg-Kirkpatrick et al., 2012)). For DIPAIRTSF and DIPAIRFFNN, we set N=4, M=12 and projected dimension D=128. Both teacher model and BERT-TINY take input with length 128. The teacher model is a customized BERT model, with a vocabulary of size 80K. BERT-TINY has a different vocab, this explains why it has the worst performance. We report the running time of the heads (measured on CPU), as #pairs $\gg$ #products $+$ #terms.

Model Settings	AUC_ROC (valid)	Delta (valid)	AUC_ROC (test)	Delta (test)	Speedup
Teacher (BERT-base)	0.955	-0%	0.957	-0%	1x
DE-FFNN	0.895	-6.3%	0.896	-6.4%	3863x
DE-COS	0.871	-8.8%	0.878	-8.3%	**5109x**
DIPAIRFFNN	0.900	-5.8%	0.904	-5.5%	2437x
DIPAIRTSF	**0.930***	**-2.6%***	**0.932***	**-2.6%***	355x

(a) Compare with Dual-Encoder based models.

Model Settings	AUC_ROC (valid)	Delta (valid)	AUC_ROC (test)	Delta (test)	Speedup
BERT-TINY	**0.933***	**-2.3%***	**0.936***	**-2.2%***	44x
DIPAIRTSF	0.930	-2.6%	0.932	-2.6%	**355x**

(b) Compare with BERT-based student model.

Table 3: Main results for Q2P-MAT data. Entries marked with * are significant (p-value < 0.05, w.r.t. the closest baseline, following the approach detailed in (Berg-Kirkpatrick et al., 2012)). For DIPAIRTSF and DIPAIRFFNN, N=4, M=8, D=256. The input to the teacher model and BERT-TINY has length 128. Query encoder and passage encoder take input with lengths 32 and 128, respectively.

#	Model Type	Head Settings	N	M	#Params in Head	AUC_ROC	Speedup
0	Teacher	-	-	-	-	0.955	1x
1	DIPAIRTSF	2-Layer	4	8	1.7M	0.930	355x
2	DIPAIRTSF	2-Layer	8	16	1.7M	0.942	98x
3	DIPAIRFFNN	$x2^7x2^7$	4	8	0.4M	0.900	2437x
4	DIPAIRFFNN	$x2^{10}x2^{10}$	4	8	4.2M	0.909	616x
5	DIPAIRFFNN	$x2^{10}x2^{10}$	8	16	7.3M	0.908	268x
6	DE-FFNN	$x2^7x2^7$	-	-	0.2M	0.895	3863x
7	DE-FFNN	$x2^{10}x2^{10}$	-	-	2.6M	0.912	754x
8	DE-FFNN	$x2^{10}x2^{10}x2^{10}x2^{10}$	-	-	4.7M	0.909	420x

Table 4: Varying the head settings in DE-FFNN, DIPAIRFFNN and DIPAIRTSF. #Params refers to the number of trainable parameters in the head. We set D=256 in DIPAIRTSF and DIPAIRFFNN. #Params is independent of N and M in DIPAIRTSF, but not in DIPAIRFFNN.

Reducing Input Sequence Length Figure 4 shows that if we reduce the input sequence length in BERT, the quality of the model drops quickly as there is not enough information available for the model to make the correct decision.

Dimension of the Projection Layer We vary the projection dimension D. Table 5 shows that AUC_ROC drops quickly when we aggressively reduce D from 256 to 16. This is expected as less information can be preserved with a smaller projection dimension. On the other hand, removing

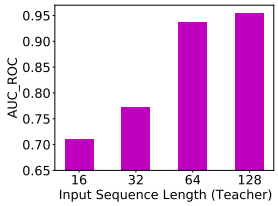


Figure 4: The effect of input sequence length.

projection layer completely leads to almost no improvement over the 256D version. This indicates that adding projection layer is a useful strategy to save both storage and running time, without hurting the model quality.

Output Dim of Projection	AUC_ROC
256D	0.930
128D	0.904
16D	0.831
No projection, 768D	0.930

Table 5: The effect of projection layer for DIPAIRTSF.

First N + M Tokens v.s. Last N + M Since our DIPAIRTSF model is end to end trained, the model should learn to push the information of the full input sequence to *arbitrarily* selected (N + M) token embeddings. To verify this intuition, we select the last (N + M) token embeddings from the dual-encoder output and compare it with the one using the first (N + M). As expected, when we fix N=4, M=8, replacing the first tokens with the last tokens only changes AUC_ROC from 0.930 to 0.925, which is almost neglectable.

Effect of Output Sequence Lengths Table 6 illustrates that for a transformer-based head, the model quality drops when we reduce the output sequence lengths ($8 \rightarrow 2$, $16 \rightarrow 2$). Here we fix D=256.

Another observation is that (N=11, M=1) is worse than any other configurations with the same value of (N+M). This might because in this Q2P-MAT data, queries are usually shorter than the passages, and we might need more token embeddings to store the information of a passage; therefore, M should greater than 1.

N	M	L	AUC_ROC
8	16	2	0.942
8	4	2	0.934
4	8	4	0.936
4	8	2	0.930
2	2	2	0.909
1	11	2	0.922
11	1	2	0.916

Table 6: The effect of output sequence lengths in DIPAIRTSF. L is the #layers in the transformer head.

5 Open Questions

DiPair has been discussed in the context of knowledge distillation in this work, but it can be trivially extended to more scenarios, as we can train it directly. The proposed framework raises several research questions.

Learning Dynamics of Our Model Recall that, in our framework, each encoder outputs its first few token embeddings as the input to the head, and we end to end to train the model to force the encoder to push the information of the input text into those outputted embeddings. However, it is unclear to us what those outputted embeddings actually learn. It would be interesting to understand the learning dynamics of our model.

Models for Online Serving In some applications, we are interested in serving the model online. Our proposed framework uses transformer-based encoders and requires to pre-compute the embeddings. As a result, it is difficult to serve our model online. It can be extremely useful to extend our framework for online use cases. Here we give a more concrete example: To score the query to document relevance online, we can usually pre-compute the embeddings of documents and index them, so using an expensive document encoder is not an issue; however, the query encoder and the head must be run online.

Extend to Non-Textual Features Another interesting situation to consider is when one side (or both sides) of the input pair is non-textual. For example, we may care about scoring a pair of (image, document), or a pair of (audio, document). Those applications require us to modify our proposed architecture to better fit non-textual features.

6 Conclusion and Future Work

In this work, we reveal the importance of customizing models for problems with pairwise/n-ary input and propose a new framework, DiPair, as an effective solution. This framework is flexible, and we can easily achieve more than 350x speedup over a BERT-based teacher model with no significant quality drop.

Acknowledgments

We would like to thank Krishna Srinivasan for his feedback and suggestions. We would also like to thank Anand Murugappan, Corinna Cortes and Greg Friedman for their support.

References

T. Berg-Kirkpatrick, D. Burkett, and D. Klein. 2012. An empirical investigation of statistical significance in NLP. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, EMNLP-CoNLL 2012, July 12-14, 2012, Jeju Island, Korea*, pages 995–1005. ACL.

S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

D. Cer, Y. Yang, S. Kong, N. Hua, N. Limtiaco, R. St. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, B. Strope, and R. Kurzweil. 2018. Universal sentence encoder for English. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 169–174, Brussels, Belgium. Association for Computational Linguistics.

M. Chidambaram, Y. Yang, D. Cer, S. Yuan, Y. Sung, B. Strope, and R. Kurzweil. 2019. Learning cross-lingual sentence representations via a multi-task dual-encoder model. In *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*, pages 250–259, Florence, Italy. Association for Computational Linguistics.

K. Clark, M. Luong, Q. V. Le, and C. D. Manning. 2020. ELECTRA: pre-training text encoders as discriminators rather than generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

A. Das, H. Yenala, M. Chinnakotla, and M. Shrivastava. 2016. Together we stand: Siamese networks for similar question retrieval. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 378–387, Berlin, Germany. Association for Computational Linguistics.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

J. Frankle and M. Carbin. 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

J. Guo, Y. Fan, Q. Ai, and W. B. Croft. 2016. A deep relevance matching model for ad-hoc retrieval. In *CIKM '16*.

J. Guo, Y. Fan, L. Pang, L. Yang, Q. Ai, H. Zamani, C. Wu, W. B. Croft, and X. Cheng. 2019. A deep look into neural ranking models for information retrieval.

S. Han, H. Mao, and W. J. Dally. 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

M. L. Henderson, R. Al-Rfou, B. Strope, Y. Sung, L. Lukács, R. Guo, S. Kumar, B. Miklos, and R. Kurzweil. 2017. Efficient natural language response suggestion for smart reply. *CoRR*, abs/1705.00652.

G. E. Hinton, O. Vinyals, and J. Dean. 2015. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531.

A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T.Weyand, M. Andreetto, and H. Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861.

B. Hu, Z. Lu, H. Li, and Q. Chen. 2014. Convolutional neural network architectures for matching natural language sentences. In *NIPS '14*.

P. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. P. Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *CIKM '13*.

S. Humeau, K. Shuster, M. Lachaux, and J. 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.

F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W.J. Dally, and K. Keutzer. 2016. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360.

X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. 2019. Tinybert: Distilling BERT for natural language understanding. *CoRR*, abs/1909.10351.

J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus.

Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. 2019. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942.

H. Li and J. Xu. 2014. *Semantic Matching in Search*. Now Publishers Inc., Hanover, MA, USA.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O.Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

S. MacAvaney, F. Maria Nardini, R. Perego, N. Tonellotto, N. Goharian, and O. Frieder. 2020. Efficient document re-ranking for transformers by precomputing term representations.

B. Mitra, F. Diaz, and N. Craswell. 2017. Learning to match using local and distributed representations of text for web search. In *WWW '17*.

L. Pang, Y. Lan, J. Guo, J. Xu, S. Wan, and X. Cheng. 2016. Text matching as image recognition. In *AAAI '16*.

J. Rao, L. Liu, Y. Tay, W. Yang, P. Shi, and J. Lin. 2019. Bridging the gap between relevance matching and semantic matching for short text similarity modeling. In *EMNLP-IJCNLP 2019*, pages 5370–5381, Hong Kong, China. Association for Computational Linguistics.

N. Reimers and I. 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.

A. Renda, J. Frankle, and M.Carbin. 2020. Comparing rewinding and fine-tuning in neural network pruning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

S. Sun, Y. Cheng, Z. Gan, and J. Liu. 2019. Patient knowledge distillation for BERT model compression. In *EMNLP-IJCNLP 2019, Hong Kong, China, November 3-7, 2019*, pages 4322–4331. Association for Computational Linguistics.

R. Tang, Y. Lu, L. Liu, L. Mou, O. Vechtomova, and J. Lin. 2019. Distilling task-specific knowledge from BERT into simple neural networks. *CoRR*, abs/1903.12136.

I. Turc, M. Chang, K. Lee, and K. Toutanova. 2019. Well-read students learn better: The impact of student initialization on knowledge distillation. *CoRR*, abs/1908.08962.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł . Kaiser, and I. Polosukhin. 2017. Attention is all you need. In *NIPS '17*.

Ran Wang, Haibo Su, Chunye Wang, Kailin Ji, and Jupeng Ding. 2019. To tune or not to tune? how about the best of both worlds? *CoRR*, abs/1907.05338.

Shuohang Wang and Jing Jiang. 2017. A compare-aggregate model for matching text sequences. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

C. Xiong, Z. Dai, J. Callan, Z. Liu, and R. Power. 2017. End-to-end neural ad-hoc ranking with kernel pooling. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, Shinjuku, Tokyo, Japan, August 7-11, 2017*, pages 55–64.

L. Yang, Q. Ai, J. Guo, and W. B. Croft. 2016. anmm: Ranking short answer texts with attention-based neural matching model. In *CIKM '16*.

Y. Yang, D. Cer, A. Ahmad, M. Guo, J. Law, N. Constant, G. Hernández Ábrego, S. Yuan, C. Tar, Y. Sung, B. Strope, and R. Kurzweil. 2019. Multilingual universal sentence encoder for semantic retrieval. *CoRR*, abs/1907.04307.

A More Information On P2T-REL

We provide a few examples from the training data to better illustrate the goal of each task.

Product One

- Title: Aurora Dragon Fantasy Mink Blanket [Weight: Medium - 5LBS,Size: Queen].
- Description: Measures 79 inch x 96 inch and will fit a Queen of Full size bed. Soft and plush. Looks great and you will love cuddling up with at night.
- Sample terms and ratings
  - size queen: 0.83
  - dragon fantasy: 0.83
  - size: 0.16
  - T-shirt: 0.

Product Two

- Title: Versace Women's Chain Reaction Chunky Sneakers - Size 37 (7).
- Description: The classic sneaker is given a haute update with experimental details-like a lightweight, chain-linked rubber sole and a riot of color and texture-for a must-have addition to your sneaker collection. Style Name:Versace Chain Reaction Sneaker (Women). Style Number: 5663881.
- Sample terms and ratings
  - sneakers: 0.91
  - leather: 0.08
  - women: 0.58
  - size 37: 0.78

The ratings are aggregated from 3 human raters.

B Derive Q2P-MAT from MS Marco Ranking

For pairwise input, creating a transfer set that roughly follows the same distribution as the training data can be very challenging (this is, however, not a problem in industrial systems as we can easily mine unlabeled data through logs). To this end, we utilize MSMARCO Passage Ranking data as it is of large scale, and we can easily create a large amount of unlabeled data. MSMARCO Passage Ranking is designed for ranking tasks, and it has 1M+ queries and 8.8M+ passages. Other popular datasets (e.g., GLUE benchmark) are relatively small, and previous distillation works often use text augmentation techniques to create transfer set.

In our work, we would like to *directly* verify the effectiveness of model distillation, so instead of using ranking metrics (a decent scoring model does not always lead to better ranking metrics), we derive a binary classification task from the MS-MARCO data,

- First, all the human-rated query to passage pairs in MSMARCO Passage Ranking data are positive. We use that part as our positive examples.
- To create relatively hard negative pairs (so that the binary classification task can be more challenging), we encode queries/passages with the universal-sentence-encoder-qa[5] (Yang et al., 2019; Chidambaram et al., 2019) and run nearest neighborhood search (some public tools are available, e.g., (Johnson et al.)) to retrieve top-30 most relevant passages for each query. We then sample pairs with dot product below 0.53 as the negative pairs. The number of negative pairs is roughly the same as the number of positive pairs.
- For the transfer set, we simply retrieve the top-50 most relevant passages (measured via dot product of the query embedding and the passage embedding) and use those query/passage pairs as the unlabeled data.

C Poly-Encoders Fails for Long Text

Compared with DiPair, Poly-Encoders (Humeau et al., 2020) has at least the following limitations,

1. It makes a strong assumption on its input pairs: One side of the input pair should be short text (e.g., less than 20 tokens).
2. It does not extend to n-ary input.
3. It can not deal with tasks beyond regression / binary-classification.

Both 2. and 3. can be implied directly from the architecture of Poly-Encoders and assumption 1 is explicitly mentioned in (Humeau et al., 2020). In this section, we experimentally show that when the assumption in 1. is violated, Poly-Encoders becomes considerably worse than DiPair.

[5] Available in https://tfhub.dev/google/universal-sentence-encoder-qa/3

We use an internal product to product similarity dataset (P2P-REL). The average length of products is about 100, and Pearson correlation between model predictions and the human ratings is our primary metric. Our teacher model is a fine-tuned BERT-base model with a customized vocabulary, and our distillation set has 182M pairs.

Model Settings	N	M	Pearson
Teacher	-	-	0.840
DIPAIRTSF	6	6	**0.826**
POLYENCODERS	1	11	0.805
DIPAIRTSF	3	3	**0.823**
POLYENCODERS	1	5	0.790

Table 7: DIPAIRTSF v.s. POLYENCODERS on P2P-REL data. We fix K=1. For fair comparison, we remove the projection layer in both methods as a projection layer is not proposed in Poly-Encoders.

Consider the fact that a product has only about 100 tokens, we believe that for longer text such as full-page documents, the gap between POLYENCODERS and DIPAIRTSF will be even larger. We leave the verification of our hypothesis as future work.