

TITE: Token-Independent Text Encoder for Information Retrieval

Ferdinand Schlatt
Friedrich-Schiller-
Universität Jena
Jena, Germany

Tim Hagen
University of Kassel
and hessian.AI
Kassel, Germany

Martin Potthast
University of Kassel,
hessian.AI, and ScaDS.AI
Kassel, Germany

Matthias Hagen
Friedrich-Schiller-
Universität Jena
Jena, Germany

Abstract

Transformer-based retrieval approaches typically use the contextualized embedding of the first input token as a dense vector representation for queries and documents. The embeddings of all other tokens are also computed but then discarded, wasting resources. In this paper, we propose the Token-Independent Text Encoder (TITE) as a more efficient modification of the backbone encoder model. Using an attention-based pooling technique, TITE iteratively reduces the sequence length of hidden states layer by layer so that the final output is already a single sequence representation vector. Our empirical analyses on the TREC 2019 and 2020 Deep Learning tracks and the BEIR benchmark show that TITE is on par in terms of effectiveness compared to standard bi-encoder retrieval models while being up to 3.3 times faster at encoding queries and documents. Our code is available at: <https://github.com/webis-de/SIGIR-25>.

CCS Concepts

• Information systems → Language models.

Keywords

Bi-Encoder, Transformer, Attention-based Pooling

ACM Reference Format:

Ferdinand Schlatt, Tim Hagen, Martin Potthast, and Matthias Hagen. 2025. TITE: Token-Independent Text Encoder for Information Retrieval. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '25)*, July 13–18, 2025, Padua, Italy. ACM, New York, NY, USA, 11 pages. <https://doi.org/10.1145/3726302.3730094>

1 Introduction

Transformer-based encoder language models [4, 12, 22] have become ubiquitous in information retrieval. Among them, the bi-encoder [56] is one of the most popular architectures. It represents queries and documents each by a single latent embedding vector and then determines a document's relevance to a query by computing the two vectors' similarity. However, as the backbone encoder model actually computes contextualized embedding vectors for each token in the input sequence (see Figure 1, left), bi-encoder models perform pooling to obtain the single vectors.

The most popular choices are mean and [CLS] pooling (i.e., averaging all tokens' contextual embeddings or only using the first token's embedding, the [CLS] token). Empirically, these strategies are similarly effective for retrieval [39, 56] so that it may not be

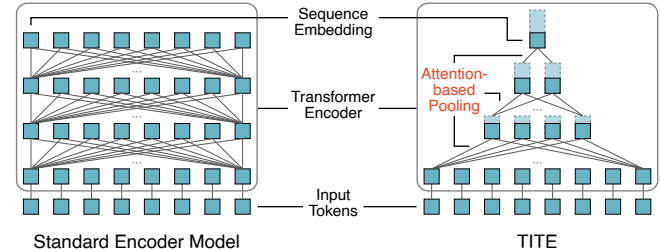


Figure 1: Comparison of a standard transformer encoder model and our new TITE model. To obtain a sequence embedding, a standard encoder model usually uses the embedding of the first token in the sequence. TITE uses attention-based pooling to reduce the sequence length of the hidden states and outputs a single vector for an input sequence. Optionally, the dimensionality of the hidden states can be increased, denoted by the transparent and dashed rectangles.

necessary to compute contextualized embeddings for all tokens, but directly computing a single sequence-level embedding suffices.

In this paper, we demonstrate that an encoder model that directly computes only a single semantic embedding vector for an input sequence can be as effective as a standard encoder model, while being substantially more efficient. For our respective new *Token-Independent Text Encoder* (TITE) we adapt and improve the Funnel-Transformer's previously introduced attention-based pooling mechanism [8] and perform pooling within transformer layers to reduce the sequence length until only a single vector remains (see Figure 1, right). To compensate for the lower number of vectors in the hidden states, we introduce the option for upscaling the dimensionality to increase the representational capacity of each vector. Additionally, we also compare several pooling strategies, locations, and arrangements to trade off efficiency vs. effectiveness and we test several different objectives for pre-training TITE.

Contrasting previous encoder models, TITE's single sequence-level output embedding vector has no direct link to the input tokens. That is, the representation is not tied to the [CLS] token or the pooled representations of multiple tokens, and, as such, is a truly token-independent sequence-level representation. Our comparison with state-of-the-art retrieval models [17, 59, 76, 78] on the TREC 2019 and 2020 Deep Learning tracks [6, 7] and the BEIR data [66] shows that TITE achieves competitive effectiveness and is up to 3.3 times faster at encoding queries and documents compared to a standard encoder model. We further find that upscaling model size improves effectiveness while upholding most of the efficiency gains. In ablation analyses, we find that the effectiveness of TITE is mainly influenced by the pre-training objective.



This work is licensed under a Creative Commons Attribution 4.0 International License. *SIGIR '25, Padua, Italy*

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1592-1/2025/07

<https://doi.org/10.1145/3726302.3730094>

2 Related Work

Transformer-based retrieval models can loosely be categorized into bi-encoder and cross-encoder models [36]. Bi-encoder models embed queries and documents separately [18, 19, 26, 29, 39, 56, 76, 77, 79, 84] and approximate a document’s relevance to the query through their embeddings’ similarity. The contextualized embeddings of the backbone encoder model allow bi-encoders to circumvent the lexical mismatch problem and to determine the semantic similarity between queries and documents. This often makes them more effective than lexical retrieval models [57].

Cross-encoders [50, 61, 63, 64, 83], on the other hand, compute the relevance score directly by passing the query and document through the encoder together. This allows them to capture complex interactions between query and document terms. It, however, also requires cross-encoders to process documents at query time, making them less efficient than bi-encoders, which can precompute an index of document embeddings. Though cross-encoders are less efficient than bi-encoders, they can be used to label training data which bi-encoders are distilled from [23]. We make use of distillation to fine-tune TITE and other baseline models.

Bi-encoder Models. Most bi-encoder architectures use a single embedding vector as the representation for queries and documents. Multi-vector bi-encoder models, which use all contextualized token embeddings of the backbone encoder model [20, 27, 30, 34, 60, 82], also exist. However, multi-vector bi-encoder models require more complex scoring functions to aggregate the similarities of each pair of query and document tokens. They further require vastly more storage for their index as each document is represented by more vectors, and they need involved multi-stage retrieval pipelines to efficiently retrieve documents [42, 46, 59]. Additionally, recent single vector bi-encoders are as effective as multi-vector models [76, 77], making the more inefficient multi-vector models less attractive.

To obtain a single semantic embedding vector for an input sequence, bi-encoders aggregate the token embeddings of the backbone encoder model. Initial work found minimal differences in effectiveness between computing the mean over token-based embeddings and using the [CLS] token as the sequence embedding [39, 56]. Since then, the design decision to use mean token pooling [26] or [CLS] pooling [18, 19, 29, 76, 77, 79] has been largely arbitrary.

However, we emphasize that there is a central difference between the two pooling strategies: In mean token pooling, a static aggregation function is applied to the token embeddings, while [CLS] pooling allows the model to learn to aggregate semantic information into a single embedding vector. Furthermore, [CLS] pooling discards the generated token embeddings and wastes the computational resources used to generate them. TITE combines the advantages of both pooling methods: A static token pooling operation is applied inside the transformer layers such that the model learns to aggregate semantic information into a single sequence embedding vector without the need for output token embeddings.

Transformer-based Encoders. We modify the backbone encoder’s architecture to obtain a single embedding vector. Transformer-based encoder models consist of several bidirectional self-attention transformer layers [69]. These layers take a sequence of vectors as input—essentially a matrix called hidden states—and output a

sequence of vectors of the same length. Attention-based pooling [8] modifies the attention mechanism to reduce the number of vectors in the output hidden states. By stacking multiple attention-based pooling layers, we iteratively reduce the sequence length and output only a single vector after the final layer.

Previous work has demonstrated that improving efficiency through pooling or compression within the attention mechanism is viable. For example, key-value compression is commonly used to make long-context decoder-only LLMs feasible [48, 81]. Other types of models like the Hourglass transformer [47, 49] and the Funnel-Transformer [8] use a pooling strategy to reduce the number of vectors. However, after pooling the representations, these types of models upsample the hidden states to the original sequence length to apply standard language modeling objectives on the token embeddings. In contrast, TITE pools the hidden states down to a single vector. Therefore, TITE’s representation is not tied to input tokens and the model cannot be pre-trained with standard token-level pre-training objectives.

Sequence-level Pre-training Objectives. Most transformer-based encoder models are pre-trained by adding noise to the input and the model is tasked to reconstruct the original sequence [4, 12, 22, 37, 54]. Based on the contextualized token embeddings of the encoder model, a linear feed-forward decoder is trained to predict the original input tokens. As TITE does not output contextualized token embeddings, we cannot use standard pre-training objectives.

Instead we use objectives designed for training sequence-level embeddings, of which several have previously been proposed. For example, the influential and original BERT encoder model pre-trained the [CLS] token’s embedding using next sentence prediction [12]. However, subsequent work discarded the next sentence prediction task as it did not improve effectiveness in downstream sequence classification tasks [22, 37, 54].

More recently, the value of training encoder models to aggregate the semantic information of an entire sequence has been re-discovered. Several encoder models have found that training the [CLS] token to aggregate semantic information improves effectiveness in semantic similarity tasks [18, 19, 26, 40, 76, 77]. For example, the Condenser model feeds hidden states from early layers and the final [CLS] token embedding into the masked language modeling head. This forces the model to aggregate semantic information in the [CLS] token to predict the masked tokens [18].

Another approach, RetroMAE, proposes combining masked language modeling with a new masked auto-encoding objective for pre-training [76, 77]. Masked auto-encoding uses an additional single layer encoder model to reconstruct an aggressively masked input sequence using the [CLS] token’s embedding vector as context information. Next to the standard method, the authors propose an enhanced variant where masking is handled by the single layer encoder model, further improving effectiveness. Finally, masked bag-of-words modeling is another alternative [40]. In masked bag-of-words modeling, a linear decoder model is fine-tuned to predict the bag-of-words distribution of a masked input representation given the [CLS] token’s embedding. We find that combining both enhanced masked auto encoding and masked bag-of-words modeling for pre-training produces the most effective TITE models.

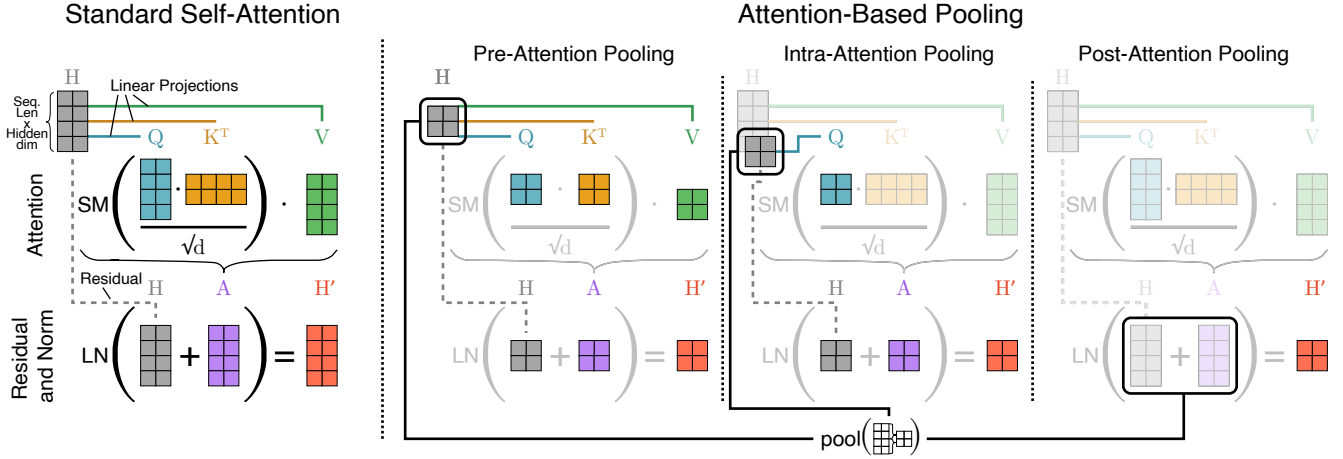


Figure 2: Comparison of standard self-attention with three attention-based pooling mechanisms. Attention-based pooling reduces the sequence length in the output hidden states H' . Pre-attention pooling applies pooling to H before the attention mechanism. Intra-attention pooling applies pooling to the query matrix Q within the attention mechanism and to the input hidden states for the residual connection. Post-attention pooling pools after the residual connection and before normalization.¹

3 Token-Independent Text Encoder

TITE is a novel type of encoder model that outputs a single latent embedding vector that captures the semantic information of the entire input sequence. To achieve this, we build upon the attention-based pooling technique introduced by Dai et al. [8]. Pooling the representations reduces the representational capacity, i.e., the number of scalar values the model can use to represent information. We compensate for this reduced capacity by upscaling the dimensionality of the hidden states. In the following, we first present the attention-based pooling technique. We then formalize how we upscale the hidden states to obtain the TITE model. The final subsection shows how we pre-train and fine-tune TITE for retrieval.

Preliminaries. A standard transformer encoder sequentially applies a token embedding layer and a stack of L self-attention transformer layers [69]. Let $t = (t_1, \dots, t_\ell)$ denote an input sequence of length ℓ where $t_i \in \{1, \dots, v\}$ is the i -th token index in the sequence and v is the size of the vocabulary. The token embedding layer then maps each token to a d -dimensional embedding vector by looking up the t_i -th row in a learned embedding matrix $E \in \mathbb{R}^{v \times d}$ for each $1 \leq i \leq \ell$. The resulting matrix, $H \in \mathbb{R}^{\ell \times d}$, is called the hidden states. Then, multiple transformer layers sequentially update H .

3.1 Attention-based Pooling

One of the central components of a transformer layer is the multi-head self-attention mechanism (MHA) [69]. It first projects the hidden states to query, key, and value matrices with which it computes dot-product attention. The original hidden states are added to the attention output using a residual connection and layer normalization (LN) is applied to obtain the updated hidden states H' :

$$H' := \text{LN}(H + \text{MHA}(q = H, k = H, v = H)).$$

Figure 2 (left) visualizes the standard self-attention mechanism.

¹ The figure is inspired by: <https://jalamar.github.io/illustrated-transformer/>

As the computational complexity $O(\ell^2 \cdot d)$ of the self-attention mechanism scales quadratically with the sequence length ℓ [69], reducing the sequence length substantially reduces the actual computational effort. Dai et al. [8] proposed the Funnel-Transformer which uses an attention-based pooling technique to reduce the sequence length of the hidden states. The central idea is to apply a convolutional mean pooling operation and aggregate the vectors of neighboring tokens into a single vector. Given a kernel size k and stride s , we define the pooling operation $\text{pool}_{(k,s)}$ as the mean vector over a strided window of vectors from the hidden states. The result is a pooled hidden states matrix, $\hat{H} \in \mathbb{R}^{\hat{\ell} \times d}$, with $\hat{\ell} \leq \ell$:

$$\text{pool}_{(k,s)} : H \mapsto \hat{H} := (\hat{h}_{i,j})_{\substack{1 \leq i \leq \hat{\ell} \\ 1 \leq j \leq d}} \text{ with } \hat{h}_{i,j} = \frac{1}{k} \sum_{n=1}^k H_{i \cdot s + n, j}.$$

Several different locations are possible for applying the pooling operation. The Funnel-Transformer pools the hidden states before passing them into the linear projection for the query matrix and also uses the pooled representations for the residual connection:

$$H' := \text{LN}(\hat{H} + \text{MHA}(q = \hat{H}, k = \hat{H}, v = H)).$$

We call this intra-attention pooling since the hidden states are pooled within the attention mechanism. Intra-attention pooling is visualized in Figure 2 (middle).

We test additional locations for pooling the hidden states as a potential means to trade-off efficiency and effectiveness. First, we also evaluate pre-attention pooling, visualized in Figure 2 (left), by applying the pooling operation prior to passing the hidden states into the attention mechanism:

$$H' := \text{LN}(\hat{H} + \text{MHA}(q = \hat{H}, k = \hat{H}, v = \hat{H})).$$

Second, we test post-attention pooling, visualized in Figure 2 (right), by applying the pooling operation after the residual connection and before the layer normalization:

$$H' := \text{LN}(\text{pool}_{(k,s)}(H + \text{MHA}(q = H, k = H, v = H))).$$

Pooling Strategies. Next to the pooling location, we investigate two further aspects to parameterize the model: (1) pooling severity, i.e., how much the sequence length is reduced by each layer, and (2) pooling arrangement, i.e., which layers of the model apply pooling. Similar to the pooling location, each aspect also has a trade-off between representational capacity and computational efficiency. Regarding pooling severity, the stronger the reduction in sequence length, the more information the model has to compress into fewer vectors at once. For pooling arrangement, if early layers in the model apply pooling, the model needs to more quickly compress low-level token information into high-level semantic information.

For severity, the Funnel-Transformer [8] uses a kernel size and stride of $k = s = 2$. This configuration halves the sequence length for each pooling layer. We additionally test $k = s = 3$, i.e., the output sequence length of a pooling layer is one third of the input sequence length to the pooling layer. While other kernel and stride configurations—such as overlapping pooling windows—would be interesting to investigate, we leave this to further work and rather investigate the effect of the arrangement of pooling layers.

In order to discuss the pooling layer arrangement, we first must determine the number of layers that need to apply pooling. Our goal with the TITE model is to output a single embedding vector for an arbitrary input sequence. We train and parameterize TITE with a maximum sequence length of 512 tokens. Therefore, we need to use nine pooling layers to reduce the sequence length to a single vector when $k = s = 2$ and six layers when $k = s = 3$.

We test two different variants for distributing the pooling layers across the twelve total layers (see Section 4.1 for further details on the model architecture). We coin the first variant late pooling and simply apply pooling to the last nine or six layers of the model, respective of the pooling severity. We call the second variant staggered pooling and distribute the pooling layers across the twelve layers. For a kernel size and stride of $k = s = 2$, layers 2–4, 6–8, and 10–12 apply pooling. For a kernel size and stride of $k = s = 3$, every second layer applies pooling, i.e., layers 2, 4, 6, 8, 10, and 12.

Finally, we emphasize that the original Funnel-Transformer [8] used a staggered strategy but only applied 3 pooling layers in total. Therefore, the output sequence length of the model is only reduced to an eighth of the original sequence length. For pre-training, the sequence length is then upsampled to the original sequence length. TITE instead reduces the sequence length to a single vector and does not upsample the hidden states, ensuring the output vector is not tied to the input tokens.

3.2 Upscaling Hidden States

Reducing the sequence length of the hidden states reduces the representational capacity of the encoder model in each layer such that the model has access to fewer vectors to capture the semantics of the sequence. To counteract this, we increase the dimensionality of the hidden states. We upscale the hidden states by increasing the dimensionality of the query Q , key K , and value V matrices within the the multi-head attention mechanism. These matrices are computed using linear projections with weights $W^x \in \mathbb{R}^{d \times d}$ and biases $\mathbf{b}^x \in \mathbb{R}^d$ for $x \in \{q, k, v\}$. We simply exchange these weights and biases for $\bar{W}^x \in \mathbb{R}^{d \times d'}$ and $\bar{\mathbf{b}}^x \in \mathbb{R}^{d'}$ with $d' > d$. The output hidden states of a (pooled) transformer layer are then $H' \in \mathbb{R}^{d' \times d'}$.

A few further modifications to the encoder architecture are necessary when applying upscaling. First, we cannot apply the residual connection within the attention mechanism as is, because the dimensionality of the output hidden states d' does not align with the dimensionality of the input hidden states d . To address this we simply add the input hidden states to the sub-matrix of the output hidden states up to the d -th column. The matrix passed into the normalization then contains the sum of the input hidden states and the self-attention output up to the d -th column and contains only the self-attention output from the $(d + 1)$ -th to the d' -th column.

Second, when pre-training encoder models (see Section 3.3 for further details on pre-training), a feed-forward decoder predicts the tokens given the final hidden states. The decoder's final layer is a linear layer, with weights $W^o \in \mathbb{R}^{d \times v}$ and biases $\mathbf{b}^o \in \mathbb{R}^v$, that maps the final hidden states to a logit distribution over the vocabulary. Previous work [55] has found that tying the decoder layer's weights W^o to the token embedding layer E , i.e., $W^o = E^T$, improves effectiveness. When upscaling the hidden states, the output hidden state's dimensionality is larger than the token embedding layer's dimensionality. To tie the weights, we replace the token embedding layer with a composed embedding matrix E' that is equal to the decoder layer's weights W^o but scaled down to the input dimensionality using a linear transformation $E' = (W^o)^T \cdot W^e$ with $W^e \in \mathbb{R}^{d \times d'}$.

Upscaling Strategies. Next to the attention mechanism, a transformer layer also includes a feed-forward network, bringing the complexity of a transformer layer to $O(\ell^2 \cdot d + \ell \cdot d^2)$ [69]. This suggests that any efficiency gains we achieve by reducing the sequence length are offset by the higher dimensionality of the hidden states. However, we do not scale the dimensionality of the hidden states linearly with the number of layers. Empirical evidence suggests that the effectiveness of language models scales with the number of parameters according to a power law [28]. Therefore, there is a diminishing return on increasing the number of parameters.

In this paper, we test a single upscaling strategy as a proof of concept and leave investigating the space of different upscaling strategies for future work. We use a staggered upscaling strategy and upscale the hidden states by one third of the original dimensionality every three layers. With 12 layers, the first three layers are d -dimensional, the next three layers $d + \frac{d}{3}$ -dimensional, then $d + \frac{2d}{3}$, and finally $2d$ -dimensional, which is also the final dimensionality of the output hidden states. We also apply the same upscaling strategy to the number of attention heads and the intermediate sizes in the feed-forward layers.

3.3 Pre-training

Previous encoder models were pre-trained using token-level pre-training objectives, for example masked language modeling [12, 22, 37] or replaced token detection [4]. This includes encoder models applying token pooling, such as the Funnel-Transformer [8], because they upscale the pooled embeddings to apply the token-level pre-training objective. As TITE outputs a single sequence-level embedding vector, we cannot apply token-level pre-training objectives. Instead, we evaluate two different sequence-level pre-training objectives: masked auto-encoding [76, 77] and masked bag-of-words modeling [40]. Figure 3 visually compares these two pre-training methods with standard masked language modeling.

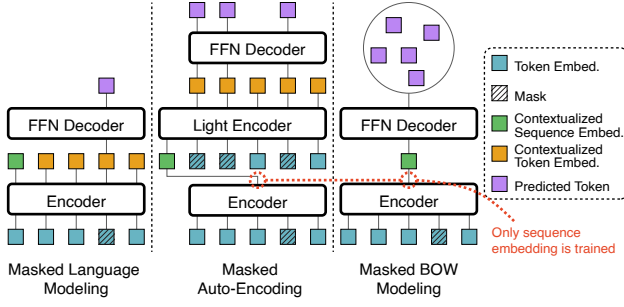


Figure 3: Comparison of masked language modeling, masked auto-encoding, and masked bag-of-words modeling. Masked language modeling needs token embeddings, whilst masked auto-encoding and masked bag-of-words modeling train the sequence embedding generated by the encoder model.

The underlying idea of masked auto-encoding is to train an additional lightweight encoder model to reconstruct the original, but aggressively masked input given the sequence-level embedding vector as context. To enable the lightweight encoder to reconstruct the input, the larger encoder model must encode the semantic information of the input sequence into the sequence-level embedding vector. In practice, the sequence-level embedding vector is prepended to the input embeddings of the original, but aggressively masked input tokens. This concatenated input is then passed into a single transformer layer and a feed-forward decoder then processes the second encoder model’s contextualized embeddings to predict the tokens of the aggressively masked input sequence. Let $X^{\text{MAE}} \in \mathbb{R}^{n \times v}$ correspond to the matrix of logit vectors output by the feed-forward decoder for masked-auto-encoding, the loss is then defined as follows:

$$\mathcal{L}_{\text{MAE}} = \frac{1}{\ell} \sum_{i=1}^{\ell} -\log \frac{\exp(X_{i,i}^{\text{MAE}})}{\sum_{j=1}^v \exp(X_{i,j}^{\text{MAE}})}.$$

We use an enhanced variant of masked auto-encoding which handles masking internally within the single-layer encoder. Therefore, the loss can be computed over all tokens in the input instead of only the tokens masked in the input. We refer to the original paper for further details [77].

In masked bag-of-words modeling, a feed-forward decoder is trained to predict the bag-of-words distribution of an input representation using only the sequence-level embedding generated by the encoder model. Let $\mathbf{x} \in \mathbb{R}^v$ equal the logit distribution over the vocabulary generated by a feed-forward decoder which receives the sequence-level embedding as input. For every token in the vocabulary, the loss computes the binary cross entropy between the predicted probability of the token being in the original unmasked sequence and whether the token was actually present in the original unmasked sequence. Let σ be the sigmoid function, $\mathbf{p} = \sigma(\mathbf{x})$ be the probability vector produced by the decoder, and $\mathbf{y}_i = 1$ if $i \in t$ and $\mathbf{y}_i = 0$ otherwise. The loss is defined as follows:

$$\mathcal{L}_{\text{BOW}} = \frac{1}{v} \sum_{i=1}^v -[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)].$$

For our overall loss we simply combine masked auto-encoding and masked bag-of-words modeling: $\mathcal{L}_{\text{pre-train}} = \mathcal{L}_{\text{MAE}} + \mathcal{L}_{\text{BOW}}$, as both pre-training methods together yielded the most effective models. Section 4.3 provides ablation tests on pre-training methods.

Lastly, we point out that masking input tokens is not necessary for masked auto-encoding and masked bag-of-words modeling. The standard token-level masked language modeling is a denoising objective, i.e., the model must reconstruct a corrupted input sequence. Masking input tokens is an integral part of the objective. Masked auto-encoding and masked bag-of-words modeling, however, are auto-encoding objectives. The encoder model must compress the input sequence such that the decoder network can reconstruct it. As such, input masking is not required for these pre-training methods and may actually be detrimental to the model’s effectiveness. Therefore, we do not apply input masking when pre-training TITE and leave investigating the effect of input masking for future work.

3.4 Fine-tuning for Retrieval

Previous work has shown that encoder models trained using masked bag-of-words modeling or masked auto-encoding can be effective zero-shot retrievers [40, 77]. However, further fine-tuning for retrieval can substantially improve effectiveness. We follow the same procedure and fine-tune TITE using distillation data.

The idea of distillation is to let a more effective but less efficient model generate relevance scores and teach a more efficient model to replicate these [23]. We follow Lassance et al. [33] and combine Margin-MSE distillation [23] and KL-divergence distillation [60]. The Margin-MSE loss function minimizes the difference between the scores of a pair of target relevance scores and a pair of predicted relevance scores. The KL-divergence distillation loss minimizes the KL-divergence between the distribution of a set of target relevance scores and a set of predicted relevance scores. We further add a contrastive InfoNCE loss using in-batch negatives and in-sample hard-negatives, as Santhanam et al. [60] found this to also improve effectiveness when combined distillation.

Formally, given a query q and a document d , let $r_{q,d} = \mathbf{q} \cdot \mathbf{d}$ denote the relevance score of the query-document pair which is the dot-product similarity of the query \mathbf{q} and document \mathbf{d} embedding vectors generated by an encoder model. Given a sequence of documents $D = (d_1, \dots, d_n)$ and a sequence of target relevance scores $Y = (y_1, \dots, y_n)$, the Margin-MSE loss, KL-divergence loss, and InfoNCE loss are defined as follows:

$$\begin{aligned} \mathcal{L}_{\text{M-MSE}} &= \frac{1}{\binom{n}{2}} \sum_{i=1}^n \sum_{j=i+1}^n ((y_{q,d_i} - y_{q,d_j}) - (r_{q,d_i} - r_{q,d_j}))^2, \\ \mathcal{L}_{\text{KL}} &= \sum_{i=1}^n \hat{y}_{q,d_i} \cdot \log \frac{\hat{y}_{q,d_i}}{\hat{r}_{q,d_i}}, \\ \mathcal{L}_{\text{InfoNCE}} &= -\log \left(\frac{\exp(r_{q,d^+})}{\sum_{d_j \in N} \exp(r_{q,d_j})} \right), \end{aligned}$$

where \hat{y} and \hat{r} are softmaxed probabilities of the respective relevance scores, d^+ is a positive sampled document, and $N = N_{\text{ib}} \cup N_{\text{is}}$ is a set of negative sampled documents consisting of in-batch negatives N_{ib} and in-sample hard-negatives N_{is} . The positive sampled document, $d^+ = \arg \max_{d_j} y_{q,d_j}$, corresponds to the document with

the highest target relevance score for the query. In-batch negatives $N_{ib} = \{d_i \mid d_i \in D_j, (q_j, D_j) \in \mathbf{B}\}$ are gathered from all other documents in the batch $\mathbf{B} = ((q_1, D_1), \dots, (q_b, D_b))$ with a batch size b . In-sample negatives $N_{is} = \{d_i \mid y_{q,d^+} - y_{q,d_i} > \theta\}$ consist of all documents in the same sample that have a threshold θ lower relevance score than the positive document. We fine-tune TITE, on the sum of these loss functions:

$$\mathcal{L}_{\text{fine-tune}} = \mathcal{L}_{\text{M-MSE}} + \mathcal{L}_{\text{KL}} + \mathcal{L}_{\text{InfoNCE}}.$$

4 Experiments and Results

We evaluate the effectiveness and efficiency of TITE to determine the impact of attention-based pooling on transformer-based retrieval models. In this section, we first provide details on our experimental setup. We then present TITE’s effectiveness using different pooling and upscaling strategies in comparison to state-of-the-art retrieval models. Ablation experiments then show the effects of different pre-training objectives. Finally, we evaluate the efficiency improvements of TITE compared to standard encoder models.

4.1 Experimental Setup

TITE Configuration. Unless otherwise specified, all models we pre-train or fine-tune are based on the BERT-base architecture [12], meaning twelve transformer layers with 768-dimensional hidden states, 12 attention heads, and 3072-dimensional linear layers in the feed-forward network. For our implementation, we use Rotary Position Embeddings (RoPE) [65] instead of absolute positional encoding to enable longer sequence lengths during inference, and we use FlashAttention-2 [9, 10] with unpadding [54, 74] to improve efficiency. Otherwise, we keep the standard architecture. For the pooling strategies, we test the four configurations described in Section 3.1. The upscaled model uses the staggered configuration described in Section 3.2 with a pooling strategy of intra-attention pooling, a kernel size and stride of $k = s = 2$, and late pooling.

Pre-training. Pre-training was done on the FineWeb-Edu corpus [53] without masking the input sequence and, following Xiao et al. [77], by masking 50% of the lightweight encoder’s inputs for masked auto-encoding. All models were pre-trained for 200k steps with a batch size of 256, and the inputs were truncated to 512 tokens for a total of about 21B tokens. The learning rate was set to 10^{-4} with 3k warm-up steps and a cosine decay schedule down to 2% of the learning rate with an AdamW optimizer [38] using the default PyTorch [52] parameters. The loss functions are described in Section 3.3. Dropout within the self-attention layers was deactivated, as we observed that they lead to unstable pre-training for TITE.

Fine-tuning. We use the MS MARCO passage dataset [1] for fine-tuning. We retrieve the top-100 passages for every query in the training dataset using ColBERTv2 [60] and obtain target relevance scores using a monoELECTRA large model [64]. We truncate queries and passages to 32 and 512 tokens, respectively. We use a batch size of 256, i.e., each batch contains 256 queries, and randomly sample 8 passages per query. All models are fine-tuned for 50k steps with a learning rate of 10^{-5} using the same optimizer and learning rate schedule as for pre-training and with the loss functions described in Section 3.4.

Evaluation. We compare the effectiveness of models using nDCG@10 on the TREC Deep Learning 2019 and 2020 tracks [6, 7] and the 14 publicly available zero-shot retrieval datasets [2, 3, 5, 11, 13, 24, 32, 45, 67, 71–73, 80] in the BEIR benchmark [66]. Queries are truncated to 32 tokens and documents to 512 tokens. For efficiency, we sample 50k queries and 50k documents from the MS MARCO passage dataset and measure the time it takes to encode them using the largest batch size that fits into GPU memory for each model.

Comparison Models. We compare TITE with BM25 [57] and several externally fine-tuned models. We also pre-train and fine-tune models to verify our pre-training and fine-tuning setup. The externally fine-tuned models are ColBERTv2 [59],² SPLADE++ [16],³ and three generic bi-encoder (Sentence-BERT) models [56]; the first⁴ is based on BERT [12], the second⁵ on DistilBERT [58], and the third on ModernBERT [74]. For Sentence-ModernBERT, we report the values from the paper, as no official checkpoint is available and fine-tuning our own model was unstable. We additionally pre-train and fine-tune our own BERT [12] and RetroMAE [77] models. Finally, we fine-tune a RetroMAE model from the officially released checkpoint.⁶ We use the indexes⁷ provided by Pyserini [35] for SPLADE++ and build our own indexes for ColBERTv2 using PLAID [59] and Sentence-BERT models using FAISS [14].

Hardware and Implementation. We pre-train and fine-tune all models on a single NVIDIA A100 GPU with 40 GB of memory. Additionally, we used the following frameworks, libraries, and tools to implement the models, run experiments, and evaluate results: ir_datasets [43], ir-measures [41], Jupyter [31], Lightning [15], Lightning IR [62], matplotlib [25], NumPy [21], pandas [51], PyTrier [44], PyTorch [52], SciPy [70], and Transformers [75].

4.2 Effectiveness Results

Table 1 reports the nDCG@10 scores on the TREC Deep Learning 2019 and 2020 tracks and the BEIR benchmark for the baseline models and our TITE models with the default configuration (intra-attention and late pooling with $k = s = 2$) with or without upscaling. We macro average the effectiveness for all sub datasets in the CQADupStack dataset into a single value and additionally report the arithmetic and geometric macro-averaged mean values over all BEIR datasets to provide an overview of the overall effectiveness. We check for statistically significant differences to our reproduced Sentence-BERT model using a two-tailed paired t-test with Holm-Bonferroni correction at a significance level of $p < 5\%$. We do not report significance for the arithmetic and geometric mean values as well as CQADupStack as these are macro-averaged values.

Model Comparison. First, we find that our reproduced Sentence-BERT and RetroMAE models achieve on par effectiveness with the officially released checkpoints, validating our pre-training and fine-tuning setups. Our Sentence-BERT model is slightly but not significantly less effective than the released checkpoint on the TREC Deep Learning 2019 and 2020 tracks. For out-of-domain retrieval on BEIR, the difference in effectiveness on average across all datasets is negligible. The results are similar when comparing our reproduced RetroMAE model to the official checkpoint.

² ColBERTv2  ³ SPLADE++  ⁴ Sentence-BERT  ⁵ Sentence-DistilBERT 



⁶ RetroMAE  ⁷ Pyserini BEIR Indexes 

Table 1: Effectiveness in nDCG@10 on TREC Deep Learning 2019 and 2020 and BEIR. S-ModernBERT scores are taken from the paper [74]. The highest scores per dataset are marked bold. Reproduced models were pre-trained and fine-tuned using our setups. † indicates statistical significance ($p < 5\%$, Holm-Bonferroni corrected) to the underlined Sentence-BERT model. CQADupStack, arithmetic mean, and geometric mean values are macro-averaged and therefore excluded from statistical testing.

Model	TREC DL		BEIR																
	2019	2020	Climate- ArguAna	CQADupStack	DBPedia	FEVER	HotpotQA	NFCorpus	NQ	Quora	SCIDOCS	TREC-COVID	SciFact	Arith. Mean	Geom. Mean				
			FEVER	FiQA	HotpotQA	NFCorpus	Quora	SciFact		Touché	Mean	Mean							
BM25	.506 [†]	.480 [†]	.397 [†]	.165 [†]	.302	.318 [†]	.651 [†]	.236 [†]	.633 [†]	.322	.305 [†]	.789 [†]	.149	.679 [†]	.595 [†]	.442[†]	.427	.379	
S-BERT (reproduced)	.700	.688	.336	.224	.319	.369	.727	.317	.574	.303	.510	.844	.146	.603	.756	.256	.449	.399	
S-BERT	.705	.726	.384 [†]	.221	.337	.385	.762 [†]	.323	.585 [†]	.315	.522 [†]	.844	.146	.606	.744	.237	.458	.407	
S-DistilBERT	.705	.699	.355 [†]	.233	.322	.375	.774 [†]	.286 [†]	.571	.298	.497 [†]	.833 [†]	.140	.596	.666 [†]	.224	.441	.391	
S-ModernBERT	–	–	.357	.236	.331	.238	.599	.288	.461	.237	.395	.859	.125	.570	.721	.208	.402	.352	
RetroMAE (reproduced)	.723	.711	.375 [†]	.242[†]	.340	.406 [†]	.737 [†]	.340 [†]	.624 [†]	.336 [†]	.539 [†]	.844	.163[†]	.663 [†]	.780	.273	.476	.428	
RetroMAE	.712	.730	.367 [†]	.240 [†]	.342	.428 [†]	.777 [†]	.343 [†]	.668 [†]	.325 [†]	.573[†]	.853[†]	.160 [†]	.638	.759	.280	.482	.432	
ColBERTv2	.732	.724	.453 [†]	.176 [†]	.359	.441[†]	.774 [†]	.346 [†]	.665 [†]	.330 [†]	.547 [†]	.851 [†]	.150	.691 [†]	.732	.257	.484	.427	
SPLADE++	.731	.720	.520[†]	.230	.334	.437 [†]	.788[†]	.347[†]	.687[†]	.347[†]	.538 [†]	.834 [†]	.159 [†]	.704[†]	.727	.247	.493	.440	
TITE (Base)	.705	.670	.391 [†]	.204 [†]	.312	.376	.699 [†]	.302	.604 [†]	.334 [†]	.484 [†]	.818 [†]	.156	.647 [†]	.691	.271	.449	.403	
TITE (Upscale)	.724	.686	.373 [†]	.209 [†]	.323	.374	.704 [†]	.298	.616 [†]	.328 [†]	.490 [†]	.827 [†]	.155	.632	.715	.275	.451	.404	

Regarding TITE, the base model without upscaling achieves marginally higher effectiveness on the 2019 and marginally lower effectiveness on the 2020 TREC DL tracks compared to our Sentence-BERT reproduction. Again, the differences are not statistically significant. For out-of-domain retrieval on BEIR, TITE is more effective on some datasets and less effective on others. The differences are significant in 8 of 13 cases, showing a fair amount of fluctuation, but on average across all datasets the effectiveness is virtually identical.

Upscaling hidden states within the TITE model leads to a minor increase in effectiveness for in-domain retrieval. The upscaled model is about 0.02 nDCG@10 more effective than the base model. For out-of-domain retrieval, upscaling the hidden states does not lead to noticeable difference in effectiveness. To our knowledge, this is the first instance of a transformer-based language model that allows increasing the dimensionality of hidden states. Our results show that this approach has merit and further research is warranted to explore its potential further.

Comparing with other more efficiency-oriented models, we find that Sentence-DistilBERT features similar effectiveness to TITE, but Sentence-ModernBERT is substantially less effective. Despite using half as many layers as a standard BERT model, Sentence-DistilBERT is more or less on par for in-domain and out-of-domain retrieval. Sentence-ModernBERT, on the other hand, is the least effective model in our comparison. Note that we were unable to fine-tune Sentence-ModernBERT using our setup due to instabilities in the fine-tuning process and rather use the values reported in the paper [74]. Future work is necessary to determine whether the effectiveness of ModernBERT can be improved.

As such, TITE achieves similar effectiveness compared to previous popular single-vector bi-encoders. Compared to more complex retrieval models like ColBERTv2 and SPLADE, TITE does not reach the same effectiveness for in and out-of-domain retrieval. ColBERTv2 and SPLADE both take advantage of the contextualized

token embedding vectors, suggesting that token-level embeddings are actually necessary to increase retrieval effectiveness. In other words, because the Sentence-BERT models use the [CLS] token’s embedding as the sequence embedding and our TITE model uses attention-based pooling to obtain a single sequence-level embedding vector they do not reach the same effectiveness as ColBERTv2 and SPLADE which use all tokens’ embeddings.

However, the comparison to RetroMAE—which uses [CLS] pooling but is specifically pre-trained for retrieval—shows single vector bi-encoder models are competitive with more complex models. RetroMAE is slightly more effective than ColBERTv2 and slightly less effective than SPLADE for both in and out-of-domain retrieval. As TITE and RetroMAE are pre-trained and fine-tuned nearly identically, the question arises whether the effectiveness difference comes from the modified encoder architecture using attention-based pooling or from the different pre-training objectives.

4.3 Ablation Experiments

TITE is as effective as a Sentence-BERT model but less effective than RetroMAE. The three models differ in their backbone encoder architecture and pre-training objectives. Sentence-BERT and RetroMAE share the same backbone encoder model, but Sentence-BERT is pre-trained using only masked language modeling, while RetroMAE is additionally pre-trained using masked auto-encoding. TITE uses attention-based pooling to increase the efficiency of the backbone encoder (see Section 4.5 for efficiency results) and outputs only a single sequence representation vector for an input sequence. As a consequence, it cannot use masked language modeling for pre-training and instead combines masked auto-encoding with masked bag-of-words modeling.

We conduct ablation experiments to determine how the pre-training objectives and backbone architectures impact effectiveness. Table 2 compares nDCG@10 on the TREC Deep Learning

Table 2: Ablation results over model architecture and pre-training objectives. We report the nDCG@10 scores on the TREC DL 2019 and 2020 tracks and arithmetic and geometric means over all datasets in the BEIR benchmark.

Model	\mathcal{L}			TREC DL		BEIR	
	MLM	MAE	BOW	2019	2020	Arith.	Geom.
S-BERT	✓	✗	✗	.700	.688	.449	.400
RetroMAE	✓	✓	✗	.723	.711	.476	.428
CLS-BERT	✗	✓	✓	.704	.674	.444	.400
TITE	✗	✓	✓	.705	.670	.449	.403
TITE	✗	✓	✗	.657	.657	.400	.353
TITE	✗	✗	✓	.660	.676	.426	.380

2019 and 2020 tracks and the BEIR benchmark for our reproduced Sentence-BERT and RetroMAE models, our TITE model, and a BERT model pre-trained using the same pre-training objectives as TITE. We call this model CLS-BERT, as only the [CLS] token’s contextualized embedding is used during pre-training. The table additionally reports the effectiveness of TITE pre-trained using only masked auto-encoding or masked bag-of-words modeling.

First and foremost, we find that CLS-BERT and TITE have virtually the same effectiveness. Pre-training only the [CLS] token and discarding the other contextualized token embeddings gives the same effectiveness as applying attention-based pooling within the encoder model to obtain only a single sequence-level embedding vector. Therefore, the effectiveness difference between TITE and RetroMAE is not due to the different encoder architecture.

Rather, the effectiveness difference is caused by the different pre-training objectives. Pre-training TITE with masked-auto-encoding and masked bag-of-words modeling leads to the same effectiveness as pre-training BERT with masked language modeling. RetroMAE improves effectiveness by combining masked language modeling with masked auto-encoding. Additionally, TITE pre-trained on both masked auto-encoding and masked bag-of-words modeling is more effective than when pre-trained with only one of the objectives. Further work on new and more advanced pre-training objectives may be able to close this small gap in effectiveness.

4.4 Pooling Parameterizations

Table 3 compares the effectiveness of TITE models using different pooling parameterizations. It reports nDCG@10 for the TREC Deep Learning 2019 and 2020 tracks and the arithmetic and geometric means for the BEIR benchmark. We compare two pooling severities ($k=s=2$ and $k=s=3$), two pooling arrangements (late and staggered pooling), and three pooling locations (pre-, intra-, and post-attention pooling). The first model (①) corresponds to the default base configuration from Table 1. We also include the upscaled model (⑦) for comparison.

First, across all pooling parameterizations we observe virtually no difference in effectiveness for out-of-domain retrieval. Across all variations of pooling severity, arrangement, location, and upscaling (①–⑦), the differences in arithmetic mean and geometric mean effectiveness are within a range of 0.008 and 0.005, respectively.

Table 3: Effectiveness in nDCG@10 on TREC Deep Learning 2019 and 2020 and BEIR (arithmetic and geometric means) for TITE models varying the kernel size and stride, pooling arrangement, pooling location, and output dimensionality. The highest scores per dataset are marked bold. Circled numbers are used to reference different TITE variants in the text.

Model Parameters						TREC DL		BEIR	
	k,s	Arr.	Loc.	Dim.		2019	2020	Arith.	Geom.
TITE	①	2	L	Intra	768	.705	.670	.449	.403
	②	2	S	Intra	768	.675	.663	.443	.397
	③	3	L	Intra	768	.683	.672	.445	.400
	④	3	S	Intra	768	.673	.669	.443	.399
	⑤	2	L	Pre	768	.686	.682	.445	.400
	⑥	2	L	Post	768	.670	.683	.446	.399
	⑦	2	L	Intra	1536	.724	.686	.451	.404

Only in-domain retrieval yields notable but nonetheless small differences. Namely, more severe pooling leads to lower effectiveness. The models using a kernel size and stride of 2 (① and ②) are more effective than their counterparts using a kernel size and stride of 3 (③ and ④). The differences for different pooling arrangements is slightly more prominent. Applying pooling earlier within a model leads to lower effectiveness. The two models using late pooling (① and ③) are more effective than their staggered pooling counterparts (② and ④). Lastly, we cannot confirm the claim made by Dai et al. [8] that pooling within the attention-mechanism is the most effective strategy. We find that pooling before, within, or after the attention mechanism does not lead to a noticeable difference in effectiveness (①, ⑤, and ⑥).

4.5 Efficiency

TITE demonstrates competitive effectiveness with state-of-the-art retrieval models (see Section 4.2). At the same time, the reduction of the sequence length using attention-based pooling substantially lowers the computational cost of TITE compared to standard encoder models. We discussed the theoretical efficiency improvements in Section 3.1. Achieving these improvements in practice, however, is non-trivial and requires ensuring high occupancy of GPU resources. To ensure this, we implement our pooling operation and the corresponding backward pass using Triton [68] and accommodate attention masking and variable sequence lengths to allow the parallel processing of multiple sequences of different lengths.

We compare TITE in terms of efficiency with a standard BERT model, a smaller DistilBERT model, a Funnel-Transformer model with the same number of parameters as the BERT model, and with a ModernBERT model. Table 4 reports the number of queries and documents from the MS MARCO passage dataset that each model can process per second. The table and our analysis covers three different attention kernels (naive eager attention, PyTorch’s official scaled dot-product attention, and FlashAttention-2) for different models where available. We also evaluate the efficiency of TITE with different pooling severity, pooling arrangement, pooling location, and upscaling configurations.

Table 4: Number of documents and queries processed per second in thousands from the MS MARCO passage dataset. The numbers in parentheses indicate the improvement versus the underlined BERT model. The circled numbers are used to reference the different TITE configurations in the text.

Model	Kernel	Queries	Documents
BERT	Eager	24.0 (0.5×)	2.0 (0.2×)
DistilBERT	Eager	47.1 (1.0×)	3.7 (0.4×)
Funnel Transformer	Eager	14.2 (0.3×)	1.3 (0.1×)
TITE (Pool Param.: ①)	Eager	69.8 (1.5×)	6.7 (0.8×)
BERT	SDPA	28.9 (0.6×)	3.2 (0.4×)
DistilBERT	SDPA	57.9 (1.2×)	6.4 (0.7×)
TITE (Pool Param.: ①)	SDPA	81.2 (1.7×)	13.4 (1.5×)
BERT	Flash	48.0	8.7
ModernBERT	Flash	41.1 (0.9×)	8.3 (1.0×)
<i>k, s Arr. Loc. Dim.</i>			
TITE	① 2 L Intra 768	Flash	89.0 (1.9×)
	② 2 S Intra 768	Flash	96.0 (2.0×)
	③ 3 L Intra 768	Flash	68.1 (1.4×)
	④ 3 S Intra 768	Flash	94.6 (2.0×)
	⑤ 2 L Pre 768	Flash	89.6 (1.9×)
	⑥ 2 L Post 768	Flash	89.0 (1.9×)
	⑦ 2 L Intra 1536	Flash	70.1 (1.5×)

Model Comparison. For eager attention, the standard TITE configuration is around 2.9 and 3.4 times faster than a standard BERT model at encoding queries and documents, respectively. The efficiency-oriented DistilBERT model is faster than BERT, but still substantially slower than our TITE model. Lastly, the comparison to the Funnel-Transformer demonstrates the difficulty of achieving high efficiency. While the Funnel-Transformer also reduces the number of operations, in practice, the standard implementation from Hugging Face [75] is substantially slower than a BERT model.

Exchanging eager attention for scaled dot-product attention improves the efficiency of all models. Note that the Funnel-Transformer does not support scaled dot-product attention, so we cannot compare it to the other models. Additionally, scaled dot-product attention is the current default implementation for a BERT model from Hugging Face. Therefore, this implementation can be considered as the standard for BERT. Our TITE model is around 2.8 and 4.2 times faster than the standard BERT model at encoding queries and documents when using scaled dot-product attention. DistilBERT is again faster than BERT, but still substantially slower than TITE.

Using FlashAttention-2 again leads to further substantial efficiency improvements. DistilBERT and BERT do not support FlashAttention-2, we therefore drop DistilBERT from our comparison and use our TITE implementation without attention-based pooling as a BERT model with FlashAttention-2. We additionally add ModernBERT to our comparison, which uses FlashAttention-2. The base TITE configuration (①) is around 1.9 and 2.4 times faster than a BERT model at encoding queries and documents, respectively, when both use FlashAttention-2. However, considering that

scaled dot-product attention is the default implementation for BERT, TITE is actually around 3.1 to 6.5 times faster than the default BERT implementation. Lastly, ModernBERT is slightly slower than BERT when using the same attention kernel because ModernBERT adds additional parameters and layers.

In summary, across all three attention kernels we observe TITE is vastly more efficient than the comparison models. Additionally, the efficiency improvements are greater for documents than for queries, due to documents being longer and because the attention mechanism’s complexity scales quadratically with the sequence length. We note that the documents in the MS MARCO passage dataset are in fact rather short, and, therefore, additional efficiency improvements are likely to be achieved on longer documents.

Pooling Parameterizations. The different pooling configurations have varying effects on the efficiency of TITE. Pooling severity has a negligible impact. In fact, for the two models with late pooling, the model with a lower kernel size and stride (①) is counterintuitively more efficient than the model with a higher kernel size and stride (③) despite compressing more tokens. This is due to the substantial impact of the pooling arrangement on efficiency. The earlier pooling is applied, the more efficient the model becomes, as is evident from the comparison of the late pooling models (① and ③) to the staggered pooling models (② and ④).

The pooling location’s impact is again less pronounced but still measurable. Applying pooling earlier in the attention layer makes the model faster. The model using pre-attention pooling (⑤) is slightly faster than the model using intra-attention pooling (①), which is in turn slightly faster than the model using post-attention pooling (⑥). Finally, upscaling the model (⑦) adds some overhead, but by reducing the sequence length at the same time as increasing the dimensionality of the hidden states TITE maintains the majority of the efficiency improvements compared to a standard encoder model without attention-based pooling.

5 Conclusion

We have introduced TITE, a novel transformer-based encoder architecture that uses attention-based pooling to reduce the sequence length of hidden states layer by layer so that the model outputs a single sequence-level representation vector. We extensively evaluated several pooling configurations and found that TITE is on par in terms of retrieval effectiveness compared to a bi-encoder model using a standard encoder architecture. At the same time, TITE is up to 3.3 times faster at encoding sequences, substantially reducing pre-training, fine-tuning, indexing, and retrieval times.

Compared to more sophisticated retrieval models, TITE is competitive but slightly less effective. Our ablation tests show that this effectiveness gap is not caused by the modified encoder architecture but is due to different pre-training objectives. The development of new pre-training objectives to improve TITE’s retrieval effectiveness is an interesting direction for future work. Additionally, we plan to explore other pooling and upscaling strategies. Lastly, as TITE is a general encoder architecture, it can be used for other tasks within and beyond the field of information retrieval. Fine-tuning a TITE-based cross-encoder for efficient re-ranking or evaluating TITE’s effectiveness for other NLP tasks thus are further interesting directions for future work.

References

- [1] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2018. MS MARCO: A Human Generated Machine Reading Comprehension Dataset. doi:10.48550/arXiv.1611.09268
- [2] Alexander Bondarenko, Maik Fröbe, Meriem Beloucif, Lukas Gienapp, Yamen Ajjour, Alexander Panchenko, Chris Biemann, Benno Stein, Henning Wachsmuth, Martin Potthast, and Matthias Hagen. 2020. Overview of Touché 2020: Argument Retrieval. In *Proceedings of CLEF 2020*. 384–395. doi:10.1007/978-3-030-58219-7_26
- [3] Vera Boteva, Demian Gholipour, Artem Sokolov, and Stefan Riezler. 2016. A Full-Text Learning to Rank Dataset for Medical Information Retrieval. In *Proceedings of ECIR 2016*. 716–722. doi:10.1007/978-3-319-30671-1_58
- [4] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *Proceedings of ICLR 2020*. 14 pages. <https://openreview.net/forum?id=r1xMH1BtvB>
- [5] Arman Cohan, Sergey Feldman, Iz Beltagy, Doug Downey, and Daniel Weld. 2020. SPECTER: Document-level Representation Learning Using Citation-informed Transformers. In *Proceedings of ACL 2020*. 2270–2282. doi:10.18653/v1/2020.acl-main.207
- [6] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. 2020. Overview of the TREC 2020 Deep Learning Track. In *Proceedings of TREC 2020 (NIST Special Publication, Vol. 1266)*. 13 pages. <https://trec.nist.gov/pubs/trec29/papers/OVERVIEW.DL.pdf>
- [7] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. 2019. Overview of the TREC 2019 Deep Learning Track. In *Proceedings of TREC 2019 (NIST Special Publication, Vol. 500–331)*. 22 pages. <https://trec.nist.gov/pubs/trec28/papers/OVERVIEW.DL.pdf>
- [8] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc Le. 2020. Funnel-Transformer: Filtering out Sequential Redundancy for Efficient Language Processing. In *Proceedings of NeurIPS 2020*. 4271–4282. <https://proceedings.neurips.cc/paper/2020/hash/2cd2915e69546904e45d4a2ac9e1652-Abstract.html>
- [9] Tri Dao. 2023. FlashAttention-2: Faster Attention with Better Parallelism and Work Partitioning. doi:10.48550/arXiv.2307.08691
- [10] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. 2022. FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. In *Proceedings of NeurIPS 2022*. 16344–16359. http://papers.nips.cc/paper_files/paper/2022/hash/67d57c32e20fd0a7a302cb81d36e40d5-Abstract-Conference.html
- [11] DataCanary, hilfialkaff, Lili Jiang, Meg Risdal, Nikhil Dandekar, and tomtung. 2017. Quora Question Pairs. <https://kaggle.com/competitions/quora-question-pairs>. Kaggle.
- [12] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of NAACL 2019*. 4171–4186. doi:10.18653/v1/N19-1423
- [13] Thomas Diggelmann, Jordan Boyd-Graber, Jannis Bulian, Massimiliano Ciaramita, and Markus Leippold. 2021. CLIMATE-FEVER: A Dataset for Verification of Real-World Climate Claims. doi:10.48550/arXiv.2012.00614
- [14] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jegou. 2024. The Faiss Library. doi:10.48550/arXiv.2401.08281
- [15] William Falcon and The PyTorch Lightning team. 2023. PyTorch Lightning. doi:10.5281/zenodo.7859091
- [16] Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. 2022. From Distillation to Hard Negative Sampling: Making Sparse Neural IR Models More Effective. In *Proceedings of SIGIR 2022*. 2353–2359. doi:10.1145/3477495.3531857
- [17] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. 2021. SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking. In *Proceedings of SIGIR 2021*. 2288–2292. doi:10.1145/3404835.3463098
- [18] Luyu Gao and Jamie Callan. 2021. Condenser: A Pre-training Architecture for Dense Retrieval. In *Proceedings of EMNLP 2021*. 981–993. doi:10.18653/v1/2021.emnlp-main.75
- [19] Luyu Gao and Jamie Callan. 2022. Unsupervised Corpus Aware Language Model Pre-training for Dense Passage Retrieval. In *Proceedings of ACL 2022*. 2843–2853. doi:10.18653/v1/2022.acl-long.203
- [20] Luyu Gao, Zhuyun Dai, and Jamie Callan. 2021. COIL: Revisit Exact Lexical Match in Information Retrieval with Contextualized Inverted List. In *Proceedings of NAACL 2021*. 3030–3042. doi:10.18653/v1/2021.naacl-main.241
- [21] 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 (2020), 357–362. doi:10.1038/s41586-020-2649-2
- [22] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. DeBERTa: Decoding-Enhanced BERT with Disentangled Attention. In *Proceedings of ICLR 2020*. 21 pages. <https://openreview.net/forum?id=XPZlaotusD>
- [23] Sebastian Hofstätter, Sophia Althammer, Michael Schröder, Mete Sertkan, and Allan Hanbury. 2021. Improving Efficient Neural Ranking Models with Cross-Architecture Knowledge Distillation. arXiv. doi:10.48550/arXiv.2010.02666
- [24] Doris Hoogeveen, Karin M. Verspoor, and Timothy Baldwin. 2015. CQADup-Stack: A Benchmark Data Set for Community Question-Answering Research. In *Proceedings of ADCS 2015*. 1–8. doi:10.1145/2838931.2838934
- [25] John D. Hunter. 2007. Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering* 9, 03 (2007), 90–95. doi:10.1109/MCSE.2007.55
- [26] Gautier Izacard, Mathilde Caron, Lucas Hosseini, Sebastian Riedel, Piotr Bojanowski, Armand Joulin, and Edouard Grave. 2022. Unsupervised Dense Information Retrieval with Contrastive Learning. *Transactions on Machine Learning Research* (2022), 21 pages. <https://openreview.net/forum?id=jKN1pXi7b0>
- [27] Ziwei Ji, Himanshu Jain, Andreas Veit, Sashank J. Reddi, Sadeep Jayasumana, Ankit Singh Rawat, Aditya Krishna Menon, Felix Yu, and Sanjiv Kumar. 2024. Efficient Document Ranking with Learnable Late Interactions. doi:10.48550/arXiv.2406.17968
- [28] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling Laws for Neural Language Models. doi:10.48550/arXiv.2001.08361 arXiv:2001.08361
- [29] 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 EMNLP 2020*. 6769–6781. doi:10.18653/v1/2020.emnlp-main.550
- [30] Omar Khattab and Matei Zaharia. 2020. ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In *Proceedings of SIGIR 2020*. 39–48. doi:10.1145/3397271.3401075
- [31] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, Carol Willing, and Jupyter Development Team. 2016. Jupyter Notebooks – A Publishing Format for Reproducible Computational Workflows. In *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. 87–90. doi:10.32323/978-1-61499-649-1-87
- [32] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics* 7 (2019), 452–466. doi:10.1162/tacl_a_00276
- [33] Carlos Lassance, Hervé Déjean, Thibault Formal, and Stéphane Clinchant. 2024. SPLADE-v3: New Baselines for SPLADE. doi:10.48550/arXiv.2403.06789
- [34] Jinhyuk Lee, Zhuyun Dai, Sai Meher Karthik Duddu, Tao Lei, Iftekhar Naim, Ming-Wei Chang, and Vincent Y. Zhao. 2024. Rethinking the Role of Token Retrieval in Multi-Vector Retrieval. In *Proceedings of NeurIPS 2024*. 22 pages. https://proceedings.neurips.cc/paper_files/paper/2023/file/31d997278ee9069d6721bc194174bb4c-Paper-Conference.pdf
- [35] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. 2021. Pyserini: An Easy-to-Use Python Toolkit to Support Replicable IR Research with Sparse and Dense Representations. In *Proceedings of SIGIR 2021*. 2356–2362. doi:10.1145/3404835.3463238
- [36] Jimmy Lin, Rodrigo Nogueira, and Andrew Yates. 2022. *Pretrained Transformers for Text Ranking: BERT and Beyond*. doi:10.1007/978-3-031-02181-7
- [37] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. doi:10.48550/arXiv.1907.11692 arXiv:1907.11692
- [38] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *Proceedings of ICLR 2019*. <https://openreview.net/forum?id=Bkg6RiCqY7>
- [39] Wenhao Lu, Jian Jiao, and Ruofei Zhang. 2020. TwinBERT: Distilling Knowledge to Twin-Structured Compressed BERT Models for Large-Scale Retrieval. In *Proceedings of CIKM 2020*. 2645–2652. doi:10.1145/3340531.3412747
- [40] Guangyuan Ma, Xing Wu, Zijia Lin, and Songlin Hu. 2024. Drop Your Decoder: Pre-training with Bag-of-Words Prediction for Dense Passage Retrieval. In *Proceedings of SIGIR 2024*. 1818–1827. doi:10.1145/3626772.3657792
- [41] Sean MacAvaney, Craig Macdonald, and Iadh Ounis. 2022. Streamlining Evaluation with *ir-measures*. In *Proceedings of ECIR 2022*. 305–310. doi:10.1007/978-3-030-99739-7_38
- [42] Sean MacAvaney and Nicola Tonello. 2024. A Reproducibility Study of PLAID. In *Proceedings of SIGIR 2024*. 1411–1419. doi:10.1145/3626772.3657856
- [43] Sean MacAvaney, Andrew Yates, Sergey Feldman, Doug Downey, Arman Cohan, and Nazli Goharian. 2021. Simplified Data Wrangling with *ir_datasets*. In *Proceedings of SIGIR 2021*. 2429–2436. doi:10.1145/3404835.3463254

- [44] Craig Macdonald, Nicola Tonellotto, Sean MacAvaney, and Iadh Ounis. 2021. PyTerrier: Declarative Experimentation in Python from BM25 to Dense Retrieval. In *Proceedings of CIKM* 2021. 4526–4533. doi:10.1145/3459637.3482013
- [45] Macedo Maia, Siegfried Handschuh, André Freitas, Brian Davis, Ross McDermott, Manel Zarrouk, and Alexandra Balahur. 2018. WWW'18 Open Challenge: Financial Opinion Mining and Question Answering. In *Companion Proceedings of WWW* 2018. 1941–1942. doi:10.1145/3184558.3192301
- [46] Franco Maria Nardini, Cosimo Rulli, and Rossano Venturini. 2024. Efficient Multi-vector Dense Retrieval with Bit Vectors. In *Proceedings of ECIR* 2024. 3–17. doi:10.1007/978-3-031-56060-6_1
- [47] Piotr Nawrot, Jan Chorowski, Adrian Lancucki, and Edoardo Maria Ponti. 2023. Efficient Transformers with Dynamic Token Pooling. In *Proceedings of ACL* 2023. 6403–6417. doi:10.18653/v1/2023.acl-long.353
- [48] Piotr Nawrot, Adrian Łancucki, Marcin Chochowski, David Tarjan, and Edoardo Ponti. 2024. Dynamic Memory Compression: Retrofitting LLMs for Accelerated Inference. In *Proceedings of ICML* 2024. 17 pages. <https://openreview.net/forum?id=tDRYrAkOB7>
- [49] Piotr Nawrot, Szymon Tworowski, Michał Tyrolski, Łukasz Kaiser, Yuhuai Wu, Christian Szegedy, and Henryk Michalewski. 2022. Hierarchical Transformers Are More Efficient Language Models. In *Findings of NAACL* 2022. 1559–1571. doi:10.18653/v1/2022.findings-naacl.117
- [50] Rodrigo Nogueira and Kyunghyun Cho. 2020. Passage Re-ranking with BERT. doi:10.48550/arXiv.1901.04085
- [51] The pandas development team. 2024. Pandas-Dev/Pandas: Pandas. Zenodo. doi:10.5281/zenodo.10957263
- [52] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Proceedings of NeurIPS* 2019. 8024–8035. https://proceedings.neurips.cc/paper_files/paper/2019/file/bd3ca288fee7f92f2bfa9f7012727740-Paper.pdf
- [53] Guilherme Penedo, Hynek Kydliček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. 2024. The FineWeb Datasets: Decanting the Web for the Finest Text Data at Scale. doi:10.48550/arXiv.2406.17557
- [54] Jacob Portes, Alexander R. Trott, Sam Havens, Daniel King, Abhinav Venigalla, Moin Nadeem, Nikhil Sardana, Daya Khudia, and Jonathan Frankle. 2023. MosaicBERT: How to Train BERT with a Lunch Money Budget. In *Workshop on Efficient Systems for Foundation Models @ ICML2023*. 15 pages. <https://openreview.net/forum?id=WHISGonzR>
- [55] Ofir Press and Lior Wolf. 2017. Using the Output Embedding to Improve Language Models. In *Proceedings of EACL* 2017. 157–163. <https://aclanthology.org/E17-2025/>
- [56] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings Using Siamese BERT-Networks. In *Proceedings of EMNLP-IJCNLP* 2019. 3980–3990. doi:10.18653/v1/D19-1410
- [57] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. 1994. Okapi at TREC-3. In *Proceedings of TREC 1994 (NIST Special Publication, Vol. 500–225)*. 109–126. <http://trec.nist.gov/pubs/trec3/paper/s/city.ps.gz>
- [58] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2020. DistilBERT, a Distilled Version of BERT: Smaller, Faster, Cheaper and Lighter. doi:10.48550/arXiv.1910.01108
- [59] Keshav Santhanam, Omar Khattab, Christopher Potts, and Matei Zaharia. 2022. PLAID: An Efficient Engine for Late Interaction Retrieval. In *Proceedings of CIKM* 2022. 1747–1756. doi:10.1145/3511808.3557325
- [60] Keshav Santhanam, Omar Khattab, Jon Saad-Falcon, Christopher Potts, and Matei Zaharia. 2022. ColBERTv2: Effective and Efficient Retrieval via Lightweight Late Interaction. In *Proceedings of NAACL-HLT* 2022. 3715–3734. doi:10.18653/v1/2022.naacl-main.272
- [61] Ferdinand Schlatt, Maik Fröbe, and Matthias Hagen. 2024. Investigating the Effects of Sparse Attention on Cross-Encoders. In *Proceedings of ECIR* 2024. 173–190. doi:10.1007/978-3-031-56027-9_11
- [62] Ferdinand Schlatt, Maik Fröbe, and Matthias Hagen. 2025. Lightning IR: Straightforward Fine-tuning and Inference of Transformer-based Language Models for Information Retrieval. In *Proceedings of WSDM* 2025. doi:10.1145/3701551.3704118
- [63] Ferdinand Schlatt, Maik Fröbe, Harrison Scells, Shengyao Zhuang, Bevan Koopman, Guido Zuccon, Benno Stein, Martin Potthast, and Matthias Hagen. 2024. Set-Encoder: Permutation-Invariant Inter-Passage Attention for Listwise Passage Re-Ranking with Cross-Encoders. In *Proceedings of ECIR* 2025. doi:10.1007/978-3-031-88711-6_1
- [64] Ferdinand Schlatt, Maik Fröbe, Harrison Scells, Shengyao Zhuang, Bevan Koopman, Guido Zuccon, Benno Stein, Martin Potthast, and Matthias Hagen. 2024. A Systematic Investigation of Distilling Large Language Models into Cross-Encoders for Passage Re-ranking. doi:10.48550/arXiv.2405.07920
- [65] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. RoFormer: Enhanced Transformer with Rotary Position Embedding. *Neurocomputing* 568 (2024), 127063. doi:10.1016/j.neucom.2023.127063
- [66] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. In *Proceedings of NeurIPS 2021 Track on Datasets and Benchmarks*. 24 pages. <https://openreview.net/forum?id=wCu6T5xFJeJ>
- [67] James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: A Large-scale Dataset for Fact Extraction and VERification. In *Proceedings of NAACL-HLT* 2018. 809–819. doi:10.18653/v1/N18-1074
- [68] Philippe Tillet, H. T. Kung, and David Cox. 2019. Triton: An Intermediate Language and Compiler for Tiled Neural Network Computations. In *Proceedings of MAPL@PLDI* 2019. 10–19. doi:10.1145/3315508.3329973
- [69] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. In *Proceedings of NeurIPS* 2017. 5998–6008. <https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html>
- [70] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, Ilhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, and Paul van Mulbregt. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17, 3 (2020), 261–272. doi:10.1038/s41592-019-0686-2
- [71] Ellen Voorhees, Tasmeer Alam, Steven Bedrick, Dina Demner-Fushman, William R. Hersh, Kyle Lo, Kirk Roberts, Ian Soboroff, and Lucy Lu Wang. 2020. TREC-COVID: Constructing a Pandemic Information Retrieval Test Collection. doi:10.48550/arXiv.2005.04474
- [72] Henning Wachsmuth, Shahbaz Syed, and Benno Stein. 2018. Retrieval of the Best Counterargument without Prior Topic Knowledge. In *Proceedings of ACL* 2018. 241–251. doi:10.18653/v1/P18-1023
- [73] David Wadden, Shanchuan Lin, Kyle Lo, Lucy Lu Wang, Madeleine van Zuylen, Arman Cohan, and Hannaneh Hajishirzi. 2020. Fact or Fiction: Verifying Scientific Claims. In *Proceedings of EMNLP* 2020. 7534–7550. doi:10.18653/v1/2020.emnlp-main.609
- [74] Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Nathan Cooper, Griffin Adams, Jeremy Howard, and Iacopo Poli. 2024. Smarter, Better, Faster, Longer: A Modern Bidirectional Encoder for Fast, Memory Efficient, and Long Context Finetuning and Inference. doi:10.48550/arXiv.2412.13663
- [75] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. HuggingFace's Transformers: State-of-the-art Natural Language Processing. doi:10.48550/arXiv.1910.03771
- [76] Shitao Xiao and Zheng Liu. 2022. RetroMAE v2: Duplex Masked Auto-Encoder For Pre-Training Retrieval-Oriented Language Models. doi:10.48550/arXiv.2211.08769
- [77] Shitao Xiao, Zheng Liu, Yingxia Shao, and Zhao Cao. 2022. RetroMAE: Pre-Training Retrieval-oriented Language Models Via Masked Auto-Encoder. In *Proceedings of EMNLP* 2022. 538–548. doi:10.18653/v1/2022.emnlp-main.35
- [78] Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. 2024. C-Pack: Packaged Resources To Advance General Chinese Embedding. doi:10.48550/arXiv.2309.07597
- [79] Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul N. Bennett, Junaid Ahmed, and Arnold Overwijk. 2021. Approximate Nearest Neighbor Negative Contrastive Learning for Dense Text Retrieval. In *Proceedings of ICLR* 2021. 16 pages. <https://openreview.net/forum?id=zeFrfqZln>
- [80] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of EMNLP* 2018. 2369–2380. doi:10.18653/v1/D18-1259
- [81] Jiayi Yuan, Hongyi Liu, Shaochen Zhong, Yu-Neng Chuang, Songchen Li, Guanchu Wang, Duy Le, Hongye Jin, Vipin Chaudhary, Zhaozhao Xu, Zirui Liu, and Xia Hu. 2024. KV Cache Compression, But What Must We Give in Return? A Comprehensive Benchmark of Long Context Capable Approaches. In *Findings of EMNLP* 2024. 4623–4648. doi:10.18653/v1/2024.findings-emnlp.266
- [82] Shunyu Zhang, Yaobo Liang, Ming Gong, Daxin Jiang, and Nan Duan. 2022. Multi-View Document Representation Learning for Open-Domain Dense Retrieval. In *Proceedings of ACL* 2022. 5990–6000. doi:10.18653/v1/2022.acl-long.414
- [83] Honglei Zhuang, Zhen Qin, Rolf Jagerman, Kai Hui, Ji Ma, Jing Lu, Jianmo Ni, Xuanhui Wang, and Michael Bendersky. 2022. RankT5: Fine-Tuning T5 for Text Ranking with Ranking Losses. doi:10.48550/arXiv.2210.10634
- [84] Shengyao Zhuang, Xueguang Ma, Bevan Koopman, Jimmy Lin, and Guido Zuccon. 2024. PromptReps: Prompting Large Language Models to Generate Dense and Sparse Representations for Zero-Shot Document Retrieval. In *Proceedings of EMNLP* 2024. 4375–4391. doi:10.18653/v1/2024.emnlp-main.250