LightToken: A Task and Model-agnostic Lightweight Token Embedding Framework for Pre-trained Language Models

Haoyu Wang
Purdue University
wang5346@purdue.edu

Ruirui Li
Amazon.com Inc
ruirul@amazon.com

Haoming Jiang
Amazon.com Inc
jhaoming@amazon.com

Zhengyang Wang
Amazon.com Inc
zhengywa@amazon.com

Xianfeng Tang
Amazon.com Inc
xianft@amazon.com

Bin Bi
Amazon.com Inc
bbi@amazon.com

Monica Cheng
Amazon.com Inc
chengxca@amazon.com

Bing Yin
Amazon.com Inc
alexbyin@amazon.com

Yaqing Wang
Purdue University
wang5075@purdue.edu

Tuo Zhao
Georgia Institute of Technology
tourzhao@gatech.edu

Jing Gao
Purdue University
jinggao@purdue.edu

ABSTRACT

Pre-trained language models (PLMs) such as BERT, RoBERTa, and DeBERTa have achieved state-of-the-art performance on various downstream tasks. The enormous sizes of PLMs hinder their deployment in resource-constrained scenarios, e.g., on edge and mobile devices. To address this issue, many model compression approaches have been proposed to reduce the number of model parameters. This paper focuses on compressing the token embedding matrices of PLMs, which typically make up a large proportion (around 20-30%) of the entire model parameters. Existing efforts to compress token embedding usually require the introduction of customized compression architectures or the optimization of model compression processes for individual downstream tasks, limiting their applicability in both model and task dimensions. To overcome these limitations and adhere to the principle of "one-for-all", we propose a lightweight token embedding framework named LightToken, which is able to produce compressed token embedding in a task and model-agnostic fashion. LightToken is generally compatible with different architectures and applicable to any downstream task. Specifically, through an integration of low-rank approximation, novel residual binary autoencoder, and a new compression loss function, LightToken can significantly improve the model compression ratio. To demonstrate the effectiveness of LightToken, we conduct comprehensive experiments on natural language understanding and question answering tasks. In particular, LightToken improves the state-of-the-art token embedding compression ratio from 5 to 25 and outperforms the existing token embedding compression approaches by 11% and 5% on GLUE and SQuAD v1.1 benchmarks, respectively.

KDD '23, August 6–10, 2023, Long Beach, CA, USA
© 2023 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0103-0/23/08.
https://doi.org/10.1145/3580305.3599416

CCS CONCEPTS

• **Computing methodologies** → **Natural language processing**.

KEYWORDS

Pre-trained Language Model, Compression

ACM Reference Format:
Haoyu Wang, Ruirui Li, Haoming Jiang, Zhengyang Wang, Xianfeng Tang, Bin Bi, Monica Cheng, Bing Yin, Yaqing Wang, Tuo Zhao, and Jing Gao. 2023. LightToken: A Task and Model-agnostic Lightweight Token Embedding Framework for Pre-trained Language Models. In *Proceedings of the 29th ACM SIGKDD Conference on KnowledgeDiscovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599416

1 INTRODUCTION

Pre-trained language models (PLMs), such as BERT [11], RoBERTa [27], and DeBERTa [16], have achieved state-of-the-art results on a wide range of natural language tasks. Examples include natural language inference [19], sentiment classification [32], and question answering [34]. These PLMs are usually very large-scale, which consist of billions of parameters; for example, BERT, RoBERTa have 110M, 123M parameters respectively. This large scale of PLMs has been a bottleneck when deploying them on resource-constrained mobile and edge devices due to hardware limitations.

To comply with the demand for lightweight models, a number of works compress PLMs to reduce their footprint through knowledge distillation [17], matrix factorization [36], pruning [53], or quantization [33]. These methods either train shallow models with fewer layers, such as DistilBERT [37], or use matrix factorization and quantization to reduce the memory consumption of each weight matrix, such as FWSVD [18] and Q8bert [49]. These compressed PLMs have been successful in reducing over 40% parameters while maintaining over 97% model performance.

Despite these progresses, there is still substantial room to achieve a much better compression ratio with less performance degradation.

PLMs typically consist of a token embedding matrix, a deep neural network with the attention mechanism, and an output layer. The token embedding matrix often occupies a significant proportion of the whole model due to the use of a large vocabulary table. For example, the token embedding matrix accounts for over 21% and 31.2% of the model size for BERT and RoBERTa respectively. Furthermore, because of differences among token frequencies, there are lots of redundancies in the token embedding matrix. Therefore, any approach that could compress token embedding matrix can be complementary to other model compression methods to achieve a higher compression ratio.

Unfortunately, very limited efforts were spent on studying the compression of the token embedding matrices, and existing methods suffer from the limitations in terms of generalizability and efficiency. Existing work [25, 51] on token embedding matrix compression leverages knowledge distillation to learn a small task-specific token embedding matrix. However, they are task-specific and thus a new compressed token embedding matrix has to be learnt for each downstream task. This could be laborious and it is difficult to reuse the compression for different tasks. Another limitation is that these approaches use knowledge distillation to guide the learning of the lightweight token embedding matrix. This process is time-consuming and expensive, for example, as noted by Zhao et al. [51], it can take 2-4 days to train on a 32 TPU cores platform, which almost takes the same time as training BERT from scratch [34].

To address the aforementioned issues and limitations, a desirable token embedding matrix compression method for PLMs should 1) be task-agnostic to generalize well for different downstream tasks, 2) be model-agnostic to be a plug-in module for different backbones, 3) be complementary to other model compression methods, and 4) have a high compression ratio while preserving model performance.

To achieve this goal, a task and model-agnostic lightweight token embedding framework for PLMs named `LightToken` is proposed in this paper. `LightToken` integrates low-rank and hashing approximation. First, we use singular value decomposition for the token embedding matrix to achieve rank-k approximation, where k is very small. Then, a residual binary autoencoder is proposed to learn the hash codes with respect to the residual matrix between the original token embedding matrix and rank-k approximation matrix. This novel integration of low-rank approximation and hashing methods could lead to significant compression with very limited performance degradation due to the following reasons. First, the low-rank approximation extracts the coherent components with a shared basis, while the residual binary autoencoder generates hash codes to approximate the incoherent components of token embedding matrix. They are complementary and jointly provide huge reduction. In addition, after the low-rank approximation, fewer hash codes are needed to maintain model accuracy, increasing the compression ratio. This is theoretically demonstrated in our analysis in Proposition 1. Furthermore, a new reconstruction loss is proposed, which could be considered as an upper bound of the widely used Euclidean distance and enable the model to focus more on the direction matching between original embeddings and compressed embeddings. It reweighs the cosine similarity and makes the loss better-conditioned than Euclidean distance.

We also propose effective compressor training recipes. Specifically, we propose to train the compressor with the pre-training objective to reduce the accuracy loss caused by compression. We fix the learned hash codes and fine-tune the decoder of the residual binary autoencoder using Wikipedia corpus. This step is efficient compared to pre-training because the number of decoder parameters is very small (a one or two hidden layers of DNN) and the epoch of fine-tuning is small (just 10 epochs). We also propose to replace the token embedding layer of the PLM with `LightToken` and fine-tune the decoder and other layers of the PLM for downstream tasks.

The contributions of the paper are summarized as follows:

- We propose a task and model-agnostic lightweight token embedding framework for PLMs by leveraging the complementary integration of low-rank and hashing approximation, and demonstrate its effectiveness both empirically and theoretically. To the best of our knowledge, this is the first attempt to leverage hashing method for learning a task and model-agnostic lightweight token embedding matrix.
- New designs and strategies are proposed to further improve the ability of the compressor. In particular, we propose a new loss function Uℓ_2 for token embedding compression, which is not only better-conditioned but also shows nice compression results.
- We conduct extensive experiments on GLUE and SQuAD v1.1 benchmarks. Results show that the proposed `LightToken` outperforms baselines significantly. The proposed `LightToken` achieves 25x compression ratio without accuracy loss, and when the compression ratio reaches 103, the accuracy loss is within 6% of the original measure.

2 RELATED WORK

State-of-the-art compression methods for pre-trained language models (PLMs) can be roughly categorized into four classes: matrix factorization, weight quantization, pruning, and knowledge distillation.

Matrix factorization (MF) uses the product of multiple small matrices to replace a large full-rank matrix. Winata et al. [47] designed an SVD-based compressed LSTM network for the question-answering task, and Acharya et al. [1] applied low-rank matrix factorization to the word embedding layer for classification model compression. MF in these methods is developed for traditional shallow language models, but not deep PLMs. Recently, Chen et al. [6] and Hsu et al. [18] proposed two variants of SVD to compress the all weight matrices of PLM. Hsu et al. [18] used fisher information as the weight of the parameters in the reconstruction loss, and Chen et al. [6] weighs the reconstructed loss by the empirical distribution of the input. ALBERT [23] factorized the token embedding layer and used the weight-sharing strategy in the pre-training stage to learn a lightweight PLM.

Weight quantization is a widely explored method for model compression, particularly in the computer vision field [9, 12, 21, 52]. The idea is to map model weights to low-precision integers and floating-point numbers. Recently, there have been many efforts toward applying this technique to PLM compression. For example, Jin et al. [21], Xiao et al. [48] and Zafrir et al. [49] proposed 8-bit quantization for BERT [11]; Tang et al. [42] further explored

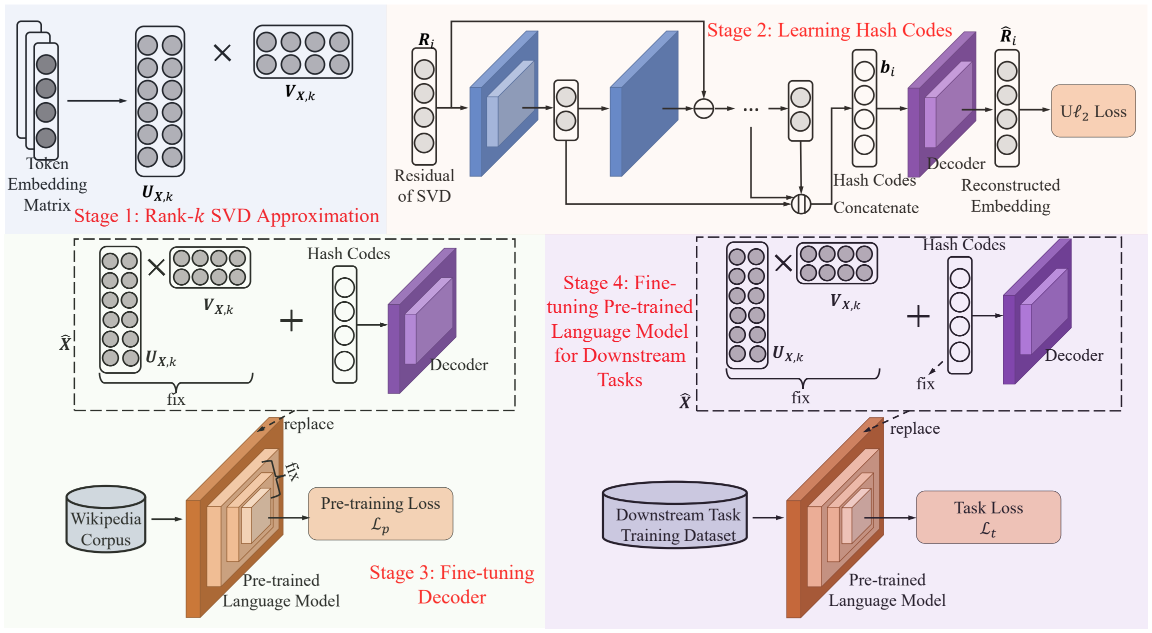

Token Embedding Matrix
$U_{X,k}$
$V_{X,k}$
Stage 1: Rank-k SVD Approximation
R_i
Residual of SVD
Stage 2: Learning Hash Codes
b_i
Hash Codes
Concatenate
Decoder
$\hat{R}_i$
Reconstructed Embedding
$U\ell_2$ Loss
$\hat{X}$
Hash Codes
fix
replace
Wikipedia Corpus
Pre-trained Language Model
Pre-training Loss $\mathcal{L}_p$
Stage 3: Fine-tuning Decoder
Stage 4: Fine-tuning Pre-trained Language Model for Downstream Tasks
Downstream Task Training Dataset
Task Loss $\mathcal{L}_t$

Figure 1: The 4-stage framework of LightToken.

using 4-bits to quantize BERT; Shen et al. [38] designed a Hessian-based mixed-precision quantization scheme for BERT; Bai et al. [2], Tian et al. [44] and Zhang et al. [50] explored how to quantize BERT into 1-bit and 2-bits. Another line of quantization is based on clustering, where multiple centroids are used to approximate vectors. For example, Shu and Nakayama [39] and Chen et al. [8] use the sum of several centroids to represent vectors.

Pruning is mainly to set redundant parameters to zero in order to learn a sparse network. Liu et al. [28] proposed a dynamic structured pruning method for efficient BERT inference; McCarley et al. [30] applied weight pruning to BERT-based question-answering models; Chen et al. [7] pruned BERT by finding a small sub-network based on lottery ticket hypothesis [13].

Knowledge distillation [17] is a compression paradigm that utilizes a trained large model (teacher model) to guide the learning of a compact, lightweight model (student model). It has been studied in various recent works, such as DistilBERT [37], TinyBERT [20], MobileBERT [41], and PKD [40], which used BERT as the teacher model to learn a more shallow or narrow student model. Zhao et al. [51] proposed a sub-token sharing vocabulary table and uses knowledge distillation to learn the new vocabulary representations. Lioutas et al. [25] combined the autoencoder and knowledge distillation to learn lightweight token representations.

Most of the existing MF, quantization, and pruning methods, however, are task-specific, and thus they cannot be used as a plug-in module. Knowledge distillation methods such as TinyBERT, MobileBERT, and PKD are model-specific and need to train a new network from scratch, which could be time-consuming. On the contrary, the proposed method is task-agnostic and can be implemented as a plug-in module to work together with many other compression methods. It is worth mentioning that task-agnostic compression is much more challenging than task-specific compression. The original token embedding matrix is very informative. Nevertheless, for task-specific compression, the compressed token embedding matrix just needs to adjust for a specific task, so extensive information can be removed or lost. In contrast, task-agnostic compression needs to preserve as much information as possible to be adapted to different downstream tasks.

Furthermore, many existing knowledge distillation methods such as TinyBERT, MobileBERT, and PKD are model-specific, relying on customized compression architectures, and require training a new network from scratch, which can be expensive. Different from them, the proposed method is model agnostic and can be applied to a variety of backbone models.

3 METHODOLOGY

Given a pre-trained language model (PLM), the token embedding, denoted as $\boldsymbol{X} \in \mathbb{R}^{d\times N}$, is a matrix with dimensions $d \times N$, where d is the dimension of each token embedding and N is the number of tokens. Typically, N is a large number, often exceeding $10K$ or even $100K$. The aim is to compress the matrix $\boldsymbol{X}$ such that the resulting compressed token embedding matrix, denoted as $\hat{\boldsymbol{X}}$ can be represented with a very small number of parameters while preserving the model performance.

3.1 Overview

In order to find $\hat{\boldsymbol{X}}$, we propose a lightweight token embedding framework named LightToken via low-rank and hashing approximation, which is shown in Fig. 1. The framework includes four stages. First, we apply singular value decomposition to the raw token embedding matrix to achieve the best rank-k (k is very small) approximation (Section 3.2). Second, the residual between SVD approximation and the raw token embedding matrix is encoded as hash codes via learning a binary autoencoder (Section 3.3). In order to make the optimization of binary autoencoder better-conditioned, we propose a new reconstruction loss function in Section 3.3.2. To further reduce the approximation error, in the third stage, we further train the compressor with the pre-training objective using a subset of Wikipedia Corpus (Section 3.4). In the fourth stage, we replace the raw token embedding matrix with the rank-k SVD, the learned hash codes, as well as the decoder and then fine-tune the decoder and later layers of PLM for downstream tasks (Section 3.5).

3.2 Rank-k SVD Approximation

A lot of existing works [3, 6, 18] have shown the singular value decomposition is powerful when compressing model weight matrices. Usually, a token embedding matrix has very few dominant singular values. Take BERT as an example, first four singular values are much larger than others as shown in Fig. 2. Therefore, we apply SVD to achieve a coarse approximation, which can be represented as

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^T \approx \boldsymbol{U}_{\boldsymbol{X},k}\boldsymbol{V}_{\boldsymbol{X},k}^T, \tag{1}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices in $\mathbb{R}^{d\times d}$ and $\mathbb{R}^{n\times n}$ respectively, and $\boldsymbol{S} \in \mathbb{R}^{d\times n}$ is a diagonal matrix. $\boldsymbol{U}_{\boldsymbol{X},k} = \boldsymbol{U}\boldsymbol{S}^{\frac{1}{2}}_{\boldsymbol{X},k}$ and $\boldsymbol{V}_{\boldsymbol{X},k} = \boldsymbol{V}\boldsymbol{S}^{\frac{1}{2}}_{\boldsymbol{X},k}$ are rank-k approximation matrices, where $\boldsymbol{S}^{\frac{1}{2}}_{\boldsymbol{X},k}$ is the square root of the first k entries of $\boldsymbol{S}$.

However, the rank-k approximation does not always work well for all models. For example, the first k singular values of RoBERTa token embedding matrix are not significantly larger than other singular values, as shown in Fig. 2. Considering the approximation error ratio $\frac{\|\boldsymbol{X}-\boldsymbol{U}_{\boldsymbol{X},k}\boldsymbol{V}_{\boldsymbol{X},k}^T\|_F}{\|\boldsymbol{X}\|_F} = \sqrt{\frac{\sum_{i=k+1}^{d}\sigma_i^2}{\sum_i \sigma_i^2}}$, where σ_i is the i-th largest singular value, the error ratio will be large if the first k singular values are not large enough, i.e. most singular values having little variance. To tackle this problem, we modify the $\boldsymbol{X}$ to amplify its first k singular values, which can be formulated as $\boldsymbol{U}\boldsymbol{S}_a\boldsymbol{V}^T$, where $\boldsymbol{S}_a$ is a diagonal matrix and $diag(\boldsymbol{S}_a)[1:k] = diag(\boldsymbol{S})[1:k]$, $diag(\boldsymbol{S}_a)[k+1:] = diag(\boldsymbol{S})[k+1:]/2$. We find replacing the token embedding matrix with $\boldsymbol{U}\boldsymbol{S}_a\boldsymbol{V}^T$ does not influence the model performance and the amplified matrix enlarges the gap between the first k singular values and remaining singular values.

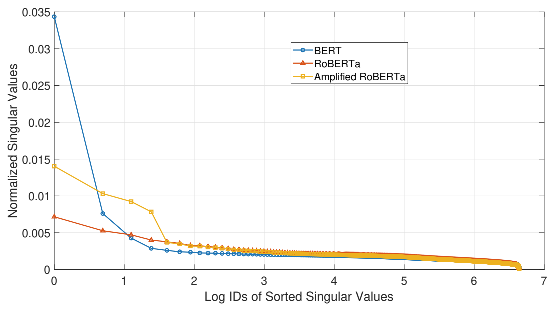


Figure 2: The curves of logarithm IDs of sorted singular values and normalized singular values.

3.3 Learning Hash Codes

3.3.1 Residual Binary Autoencoder. Although the SVD provides a coarse approximation of $\boldsymbol{X}$, we find that it loses a great deal of information of token embeddings, which leads to a significant performance drop in experiments. To alleviate performance drop and improve the approximation, we use binary hash codes to encode the residual of SVD, which is denoted as $\boldsymbol{R} = \boldsymbol{X} - \boldsymbol{U}_{\boldsymbol{X},k}\boldsymbol{V}_{\boldsymbol{X},k}^T$. Here, complementary to SVD which is a linear compression method, we design a non-linear binary autoencoder to learn the hash codes. The binary autoencoder encodes the input residual $\boldsymbol{R}$ as binary codes and then takes advantage of a non-linear decoder to reconstruct input. Specifically, the binary autoencoder can be formulated as

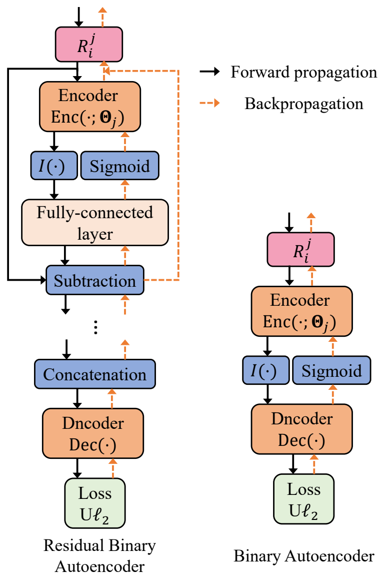


Figure 3: The framework of the Residual Binary Autoencoder and a simple Binary Autoencoder..

$$\boldsymbol{b_i} = \mathrm{Enc}(\boldsymbol{R}_i), \hat{\boldsymbol{R}}_i = \mathrm{Dec}(\boldsymbol{b}_i), \tag{2}$$

where Enc(·) is the encoder, Dec(·) is the decoder, $\boldsymbol{b_i} \in \{0,1\}^{d_b}$, d_b is the dimension of hash codes, and the subscript i represents the i-th token. However, the encoded hash codes $\boldsymbol{b}_i$ are discrete, which is non-differentiable. Therefore, it is difficult to learn the hash codes in an end-to-end manner directly. To solve this problem, following Bengio et al. [4], we apply the Straight Through estimator to estimate the gradient, i.e. using binary codes in the forward propagation and employing tempered sigmoid to approximate binary codes during backward propagation, which can be formulated as

$$\boldsymbol{e_i} = \mathrm{Enc}(\boldsymbol{R}_i), \tag{3}$$
$$\boldsymbol{b_i} = \mathrm{Sigmoid}(\boldsymbol{e_i}/\tau) + \mathrm{Sg}(\boldsymbol{I}(\boldsymbol{e_i} > 0.5) - \mathrm{Sigmoid}(\boldsymbol{e_i}/\tau)), \tag{4}$$
$$\hat{\boldsymbol{R}}_{\boldsymbol{i}} = \mathrm{Dec}(\boldsymbol{b}_i), \tag{5}$$

where Sg(·) represents the stop gradient operation, $\boldsymbol{I}(x) = 0$ if $x < 0.5$, otherwise $\boldsymbol{I}(x) = 1$, and τ is a positive scalar.

In this way, we can train the binary autoencoder with popular optimizers such as SGD, Adam, etc. However, the tempered sigmoid is a biased estimation and it will be saturating after training multiple epochs [43], which makes the model converge to a subpar local optimum. We show the differences between the residual binary autoencoder and the simple binary autoencoder in Fig. 3. To solve this problem, we propose a novel residual binary autoencoder network, which learns the hash codes in a progressive manner with residual connections. Specifically, let $\boldsymbol{R}_i^j$ be the input of j-th encoder ($j = 0, ..., m-1$), and then the recursive expressions can be represented as

$$\boldsymbol{e}_i^j = \mathrm{Enc}(\boldsymbol{R}_i^j; \Theta_j), \tag{6}$$

$$\boldsymbol{b}_i^j = \mathrm{Sigmoid}(\boldsymbol{e}_i^j/\tau) + \mathrm{Sg}(\boldsymbol{I}(\boldsymbol{e}_i^j > 0.5) - \mathrm{Sigmoid}(\boldsymbol{e}_i^j/\tau)), \tag{7}$$

$$\boldsymbol{R}_i^{j+1} = \boldsymbol{R}_i^j - \mathrm{FC}(\boldsymbol{b}_i^j), \tag{8}$$

where $\mathrm{FC}(\cdot)$ is a fully-connected layer to transform the dimension of $\boldsymbol{b}_i^j$ the same as that of $\boldsymbol{R}_i^j$, and $\boldsymbol{R}_i^0 = \boldsymbol{R}_i$. The final hashing codes are the concatenation of $\boldsymbol{b}_i^j$, and the decoder decodes the hash codes, which can be formulated as

$$\boldsymbol{b}_i = \boldsymbol{b}_i^0 \| \boldsymbol{b}_i^1 \| ... \| \boldsymbol{b}_i^{m-1}, \hat{\boldsymbol{R}}_i = \mathrm{Dec}(\boldsymbol{b}_i), \tag{9}$$

where $\|$ represents the concatenation operation. In this way, the residual connections enable the gradient to be more stable and helps the autoencoder learn better binary representation. And we just need to store the learned hash codes $\boldsymbol{B}$ and the decoder $\mathrm{Dec}(\cdot)$. To achieve a high compression ratio, the architecture can be as lightweight as possible, such as modeled by a fully-connected network with only one or two hidden layers.

3.3.2 Loss Function. The reconstructed token embedding can be represented as $\hat{\boldsymbol{X}} = \hat{\boldsymbol{R}} + \boldsymbol{U}_{\boldsymbol{X},k}\boldsymbol{V}_{\boldsymbol{X},k}^T$, and existing works usually use Euclidean distance to calculate the loss function, i.e.

$$\min_{\Theta} \|\boldsymbol{X} - \hat{\boldsymbol{X}}\|_F^2, \tag{10}$$

where Θ is the parameter of residual binary autoencoder. However, we find although using the loss function based on direct Euclidean distance can achieve good performance on most tasks, it shows poor performance on grammatical acceptability tasks, such as on CoLA [46], or on small training dataset like RTE [5, 10, 14, 15], which are shown in experiments. The potential reason is that the Euclidean distance does not pay enough attention to the angle between original embeddings and the recontructions. Specifically, the Euclidean distance can be decomposed as

$$\|\boldsymbol{X}_i - \hat{\boldsymbol{X}}_i\|_2^2 = \|\boldsymbol{X}_i - \hat{\boldsymbol{X}}_i^{\|}\|_2^2 + \|\boldsymbol{X}_i - \hat{\boldsymbol{X}}_i^{\perp}\|_2^2, \tag{11}$$

where $\hat{\boldsymbol{X}}_i^{\|} = \frac{\langle \boldsymbol{X}_i, \hat{\boldsymbol{X}}_i \rangle}{\langle \boldsymbol{X}_i, \boldsymbol{X}_i \rangle}\boldsymbol{X}_i$ is the parallel projection, and $\hat{\boldsymbol{X}}_i^{\perp} = \hat{\boldsymbol{X}}_i - \frac{\langle \boldsymbol{X}_i, \hat{\boldsymbol{X}}_i \rangle}{\langle \boldsymbol{X}_i, \boldsymbol{X}_i \rangle}\boldsymbol{X}_i$ is the orthogonal projection. After simplifying the second term, we have

$$\|\boldsymbol{X}_i - \hat{\boldsymbol{X}}_i^{\perp}\|_2^2 = \|\hat{\boldsymbol{X}}_i\|_2^2\left(1 - \left(\frac{\langle \boldsymbol{X}_i, \hat{\boldsymbol{X}}_i \rangle}{\|\boldsymbol{X}_i\|\|\hat{\boldsymbol{X}}_i\|}\right)^2\right). \tag{12}$$

We can find Eqn. 12 enforces the square of the cosine similarity between $\boldsymbol{X}_i$ and $\hat{\boldsymbol{X}}_i$ to be equal to 1. However, the cosine similarity tending to 1 or -1 can both decrease this loss term, which provides too much freedom to learn the autoencoder parameters and may confuse/mislead the approximation learning. To solve this problem, we replace this orthogonal projection term with a tight upper bound, which can be represented as

$$\|\hat{\boldsymbol{X}}_i\|_2^2\left(1 - \left(\frac{\langle \boldsymbol{X}_i, \hat{\boldsymbol{X}}_i \rangle}{\|\boldsymbol{X}_i\|\|\hat{\boldsymbol{X}}_i\|}\right)^2\right) \leq 2\|\hat{\boldsymbol{X}}_i\|_2^2\left(1 - \frac{\langle \boldsymbol{X}_i, \hat{\boldsymbol{X}}_i \rangle}{\|\boldsymbol{X}_i\|\|\hat{\boldsymbol{X}}_i\|}\right).$$

The upper bound enables the cosine similarity to be close to 1 directly and two sides of the inequality are equal if and only if $\frac{\langle \boldsymbol{X}_i, \hat{\boldsymbol{X}}_i \rangle}{\|\boldsymbol{X}_i\|\|\hat{\boldsymbol{X}}_i\|} = 1$. Therefore, the final loss function Uℓ_2 is

$$\sum_i \|\boldsymbol{X}_i - \hat{\boldsymbol{X}}_i^{\|}\|_2^2 + 2\|\hat{\boldsymbol{X}}_i\|_2^2\left(1 - \frac{\langle \boldsymbol{X}_i, \hat{\boldsymbol{X}}_i \rangle}{\|\boldsymbol{X}_i\|\|\hat{\boldsymbol{X}}_i\|}\right). \tag{13}$$

Furthermore, we can rewrite Uℓ_2 as

$$\sum_i \|\boldsymbol{X}_i - \hat{\boldsymbol{X}}_i\|_2^2 + \|\hat{\boldsymbol{X}}_i\|_2^2\left(1 - \frac{\langle \boldsymbol{X}_i, \hat{\boldsymbol{X}}_i \rangle}{\|\boldsymbol{X}_i\|\|\hat{\boldsymbol{X}}_i\|}\right)^2. \tag{14}$$

Therefore, the proposed Uℓ_2 loss can also be interpreted as adding a weighted cosine similarity regularizer in the Euclidean distance.

3.4 Training Compressor with Pre-training Objective

To further close the gap between the lightweight token embedding and raw token embedding, we fine-tune the learned decoder on the subset of Wikipedia corpus.

Formally, we replace the token embedding matrix of the PLM with the learned hash codes and decoder, and denote the parameters of the decoder as $\Theta_{\boldsymbol{D}}$, the parameters of the PLM except token embedding as Θ_{PLM}. Then we fix the hash codes $\boldsymbol{B}$, Θ_{PLM}, and only update $\Theta_{\boldsymbol{D}}$ with the pre-training loss $\mathcal{L}_p$ (masked language modeling loss), which can be represented as

$$\min_{\Theta_{\boldsymbol{D}}} \mathcal{L}_p(\mathfrak{X}; \boldsymbol{B}, \Theta_{\boldsymbol{D}}, \Theta_{PLM}), \tag{15}$$

where $\mathfrak{X}$ represents Wikipedia corpus data. There are three advantages to fine-tune the decoder. First, it helps preserve semantic information in the reconstructed embeddings. which is verified in Section 5.3.2. Second, fine-tuning the decoder is very efficient due to its simple architecture. Third, it is task and model-agnostic. It is fine-tuned with pre-training loss and does not rely on a specific PLM design.

3.5 Fine-tuning PLM for Downstream Tasks

To apply the PLM and lightweight token embedding for different downstream tasks, we fine-tune the model with task-specific loss functions. However, different from traditional task-specific fine-tuning, we fix the hash codes and only update Θ_{PLM} and $\Theta_{\boldsymbol{D}}$, which can be formulated as

$$\min_{\Theta_{PLM}, \Theta_{\boldsymbol{D}}} \mathcal{L}_t(\mathfrak{X}_t; \boldsymbol{B}, \Theta_{\boldsymbol{D}}, \Theta_{PLM}), \tag{16}$$

where $\mathfrak{X}_t$ is the downstream task training data.

4 THEORETICAL ANALYSIS

In this section, we show that hashing can benefit from rank-k SVD approximation, i.e. the number of hash codes to be required can be reduced by conducting rank-k SVD approximation. We prove it theoretically in Proposition 1 based on Lemma 1. In Lemma 1 and Proposition 1, we assume the token embedding satisfies a multivariate Gaussian distribution. We visualize the 200th and 600th dimension of BERT token embedding matrix in Fig. 4 to show the reasonableness of the assumption.

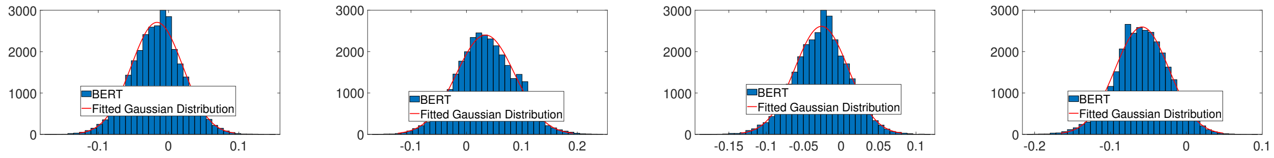


Figure 4: Histograms of four random sampled dimensions of BERT representation and curves of fitted Gaussian distribution PDF.

Lemma 1. *Assume we are given a set of N i.i.d. samples $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_N] \in \mathbb{R}^{d\times N}$ from a multivariate Gaussian distribution. We encode the vector $\boldsymbol{x}$ such that the encoded representation can be used to recover $\boldsymbol{x}$ given a distortion ϵ^2, i.e. $\mathbb{E}[\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|^2] \le \epsilon^2$. Then the average number of bits needed is*

$$L(X) = \frac{N+d}{2N}\log\left(\boldsymbol{I} + \frac{d}{\epsilon^2 N}\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^T\right) + \frac{d}{2N}\log\left(1 + \frac{\boldsymbol{\mu}^T\boldsymbol{\mu}}{\epsilon^2}\right), \quad (17)$$

where $\boldsymbol{\mu} = \frac{1}{N}\boldsymbol{X}\boldsymbol{1}_{N\times 1}$, and $\bar{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{\mu}\boldsymbol{1}^T_{N\times 1}$. If the $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_N] \in \mathbb{R}^d$ from a subspace, i.e., a degenerate Gaussian, Eqn. 18 is an upper bound of the code length [29].

Proposition 1. *Assume each row of $\boldsymbol{X}$ is sampled from a multivariate Gaussian distribution and $\frac{1}{N}\boldsymbol{X}\boldsymbol{1}_{N\times 1} = \boldsymbol{0}$. The singular value decomposition of $\boldsymbol{X}$ is $\boldsymbol{X} = \boldsymbol{U}\Lambda\boldsymbol{V}^T$. The k approximate matrix $\boldsymbol{X}_k = \boldsymbol{U}\Lambda_k\boldsymbol{V}^T$, where Λ_k is the same matrix as Λ except that it contains only the k largest singular values. Then we have $L(\boldsymbol{X} - \boldsymbol{X}_k) < L(\boldsymbol{X})$ when $\frac{\Pi_{i=1}^d(1+\frac{d}{N\epsilon^2}\sigma_i^2(\boldsymbol{X}))}{(1+\frac{d}{N\epsilon^2}\sigma_{k+1}^2(\boldsymbol{X}))^d} > (1 + \frac{d}{N^2\epsilon^2}\sigma_1^2(\boldsymbol{X}))^{\frac{N}{N+d}}$, where $\sigma_i(\boldsymbol{X})$ is the i-th largest singular value of $\boldsymbol{X}$.*

5 EXPERIMENT

In this section, we evaluate the proposed `LightToken` and answer the following questions:

RQ1 How does `LightToken` perform compared to state-of-the-art token compression baselines?
RQ2 What is the role of each module of `LightToken` in model performance improvements respectively?
RQ3 How does the performance change with varying compression ratios and ranks of SVD approximation?
RQ4 Can the proposed `LightToken` preserve semantic information of original tokens?

5.1 Datasets and Experiment Settings

5.1.1 Datasets. We conduct experiments on benchmark GLUE [45] and SQuAD 1.1 [35] following Chen et al. [6], Hsu et al. [18]. More specifically, we adopt 8 tasks from GLUE for evaluation, including MRPC, SST-2, RTE, STS-B, QQP, QNLI, MNLI, and CoLA. Details of tasks are shown in Table 3 in the appendix. The performance is reported on the development sets following [6, 18].

5.1.2 Baselines. We adopt four state-of-the-art baselines:

- SVD is a classical and powerful method for compression. It is the best rank-k approximation with respect to Euclidean distance.
- DSVD [3] is a variant of SVD. It adds cosine similarity as a regularizer and achieves state-of-the-art performance on token compression task.
- IRVQ [26] is an improved residual quantization method, which consists of multiple codebooks and takes advantage of the residual mechanism to learn the quantized representation. We implement IRVQ based on Faiss [22].
- DCQ [8, 39] is a state-of-the-art method to learn compositional representation for embedding compression in an end-to-end manner.

We compare all methods under a high compression ratio as 25. To test the generalization of `LightToken`, we apply the proposed mehtod to three popular backbones: 1) BERT-base [11], 2) RoBERTa-base [27], and 3) DistilBERT [37]. BERT and RoBERTa are used to represent BERT-base and RoBERTa-base respectively for short. We show the implementation details in the appendix due to limited space.

5.2 Performance Comparison

In this section, we report the performance of baselines and the proposed `LightToken` in Table 2 and Table 1 to answer RQ1.

Table 2 and Table 1 show that the proposed `LightToken` outperforms all the state-of-the-art baselines on GLUE and SQuAD v1.1 datasets. Under the compression ratio of 25, all baselines suffer from significant performance drops except the proposed method. The proposed `LightToken` nearly preserve the performance of full model to achieve lossless compression. The performance of baseline methods drops by at least 14.6%, 8.1%, and 9.0% with BERT, RoBERTa, and DistilBERT respectively on GLUE with respect to average performance. However, the proposed `LightToken` maintains PLMs' accuracy by taking advantage of the non-linear architecture and the new loss function. The experimental results show that the proposed method can not only be used as a plug-in component for PLMs like BERT and RoBERTa but also be use to further reduce the parameters of the compressed model like DistilBERT to improve the efficiency.

We observe that the methods SVD and DSVD with RoBERTa suffers from more than 20% performance drop, which is larger than those with BERT and DistilBERT. Such an observation illustrates that it is difficult to make single low-rank approximation work with RoBERTa and supports the motivation of of amplifying singular values in this scenario.

5.3 Ablation Study

5.3.1 Effectiveness of the Proposed Uℓ_2 Loss. In this section, we conduct ablation studies to answer RQ2 regarding the Uℓ_2 loss. To clearly validate the role of the proposed Uℓ_2 loss in `LightToken`, we design two ablation studies: 1) comparing `LightToken` with `LightToken` w/o Uℓ_2 and 2) comparing `LightToken` w/o WikiFT and `LightToken` w/o Uℓ_2 and WikiFT. The results are also summarized in Table 2 and Table 1. The ablation studies confirm that

Table 1: Performance comparison on GLUE dataset. "CR" represents token emnedding compression ratio. "AVG" is the average performance of the 8 tasks. Results of * and ♡ are taken from [37] and [24] respectively. The highest scores of compression methods per category are in bold.

Method	CR	MRPC	SST-2	RTE	STS-B	QQP	QNLI	MNLI	CoLA	AVG
BERT*	1	88.6	92.7	69.3	89.0	89.6	91.8	86.7	56.3	83.0
+SVD	25	81.1	79.4	54.2	73.1	79.8	81.2	73.6	7.7	66.3
+DSVD	25	81.5	81.1	60.0	78.2	83.4	84.5	76.4	14.5	70.0
+IRVQ	25	83.5	81.1	56.7	82.1	84.2	84.5	78.3	16.4	70.9
+DCQ	25	83.2	79.1	56.7	78.4	83.9	84.2	76.3	22.6	70.6
+LightToken (ours)	25	**90.9**	**92.2**	**71.5**	87.9	87.8	**91.2**	**84.0**	**57.6**	**82.9**
w/o Uℓ_2 w/o WikiFT	25	86.3	88.5	65.7	86.8	86.2	88.3	82.6	23.2	76.0
w/o WikiFT	25	88.2	89.8	64.6	87.4	86.8	90.4	84.2	33.9	78.2
w/o Uℓ_2	25	89.6	91.4	68.6	**88.2**	**87.9**	91.1	83.5	55.5	82.0
RoBERTa♡	1	91.7	94.5	73.7	89.1	88.8	90.8	87.2	64.8	85.1
+SVD	25	83.2	82.8	52.4	70.3	83.2	77.4	74.5	0.0	65.5
+DSVD	25	83.1	81.9	52.7	77.1	82.7	83.1	75.2	0.0	67.0
+IRVQ	25	87.2	86.9	59.6	84.5	85.5	86.6	81.5	19.0	73.9
+DCQ	25	88.8	92.1	61.7	85.2	87.2	90.5	85.4	34.9	78.2
+LightToken (ours)	25	**91.2**	**93.6**	**72.2**	**89.2**	**88.0**	91.8	**86.6**	**61.8**	**84.3**
w/o Uℓ_2 w/o WikiFT	25	86.5	90.3	58.8	86.8	87.4	90.1	84.6	8.1	74.1
w/o WikiFT	25	88.2	90.9	60.3	87.7	87.8	90.9	85.9	13.4	75.6
w/o Uℓ_2	25	91.0	93.2	70.0	88.6	**88.0**	**92.3**	86.5	57.0	83.3
DistilBERT*	1	87.5	91.3	59.9	86.9	88.5	89.2	82.2	51.3	79.6
+SVD	25	80.8	78.4	58.1	69.9	81.7	81.4	70.6	9.4	66.3
+DSVD	25	82.7	79.2	59.2	75.2	82.7	82.9	73.7	9.2	68.1
+IRVQ	25	85.1	84.3	53.8	80.2	83.7	84.2	75.6	15.0	70.2
+DCQ	25	85.1	83.5	58.5	79.0	84.5	83.3	77.9	27.0	72.4
+LightToken (ours)	25	**90.1**	**89.9**	**66.1**	**85.9**	86.5	88.4	81.6	**51.7**	**80.0**
w/o Uℓ_2 w/o WikiFT	25	87.1	86.8	63.5	84.8	85.0	86.1	80.0	19.7	74.1
w/o WikiFT	25	86.5	86.9	63.9	84.9	85.7	86.4	80.4	25.4	75.0
w/o Uℓ_2	25	89.2	89.2	63.5	85.7	**86.6**	**88.5**	**81.8**	48.1	79.1

the proposed Uℓ_2 loss helps the model achieve better performance. Take Table 1 as an example, compared to LightToken w/o Uℓ_2, the proposed LightToken achieves 1.1%, 1.2%, and 1.1% performance improvements with BERT, RoBERTa and DistilBERT as backbones respectively in term of the average accuracy over the 8 tasks; compared to LightToken w/o Uℓ_2 and WikiFT, LightToken w/o WikiFT shows 2.9%, 2.0%, and 1.2% improvements with BERT, RoBERTa and DistilBERT as backbones respectively in term of the average accuracy over the 8 GLUE tasks. In particular, we observe that the proposed loss function significantly boosts the model performance with BERT, RoBERTa, and DistilBERT on CoLA, achieving 25%, 37%, and 18% improvement in average respectively. This demonstrates the effectiveness of the proposed loss function, especially when dealing with grammar-preservation involved tasks.

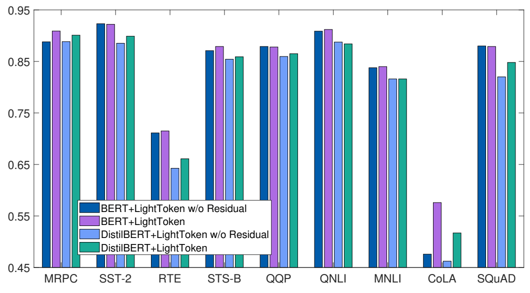


Figure 5: Performance of LightToken with or without the proposed residual connection. The y-axis represents the evaluation metric values corresponding to each task.

5.3.2 *Effectiveness of Fine-tuning the Decoder.* In this section, we conduct analysis to answer RQ2 regarding fine-tuning the decoder. We design two ablation studies including 1) comparing LightToken and LightToken w/o WikiFT and 2) comparing LightToken w/o Uℓ_2 with LightToken w/o Uℓ_2 and WikiFT to show the effectiveness of fine-tuning the decoder. We report results in Table 2 and Table 1. The experimental results show that fine-tuning the decoder can improve model performance significantly. The average performance improvements of LightToken versus LightToken w/o WikiFT and LightToken w/o Uℓ_2 versus LightToken w/o Uℓ_2 and WikiFT with respect to the three backbones on GLUE and SQuAD are 8.5% and 2.0% respectively. Specifically, LightToken brings improvements around 11.5% compared to LightToken w/o WikiFT in term of the average accuracy over 8 tasks on GLUE dataset. The results show

Table 2: Performance comparison on SQuAD v1.1 dataset. "CR" represents the token embedding compression ratio. Results of * and ♡ are taken from [37] and [16] respectively. The highest scores of compression methods per category are in bold.

Method	CR	SQuAD v1.1	
		Exact Match (EM)	F1
BERT*	1	81.2	88.5
+SVD	25	64.5	74.8
+DSVD	25	67.3	77.5
+IRVQ	25	72.7	82.0
+DCQ	25	68.8	78.8
+LightToken (ours)	25	**80.3**	**87.9**
w/o Uℓ_2 w/o WikiFT	25	79.8	87.5
w/o WikiFT	25	79.8	87.3
w/o Uℓ_2	25	80.1	87.8
RoBERTa♡	1	84.8	91.1
+SVD	25	75.1	83.2
+DSVD	25	75.9	83.7
+IRVQ	25	79.8	87.4
+DCQ	25	83.3	90.1
+LightToken (ours)	25	85.2	**91.6**
w/o Uℓ_2 w/o WikiFT	25	83.3	90.2
w/o WikiFT	25	83.0	89.6
w/o Uℓ_2	25	**85.3**	**91.6**
DistilBERT*	1	77.7	85.8
+SVD	25	56.2	68.2
+DSVD	25	59.5	70.8
+IRVQ	25	59.9	71.7
+DCQ	25	68.0	78.2
+LightToken (ours)	25	**76.4**	**84.8**
w/o Uℓ_2 w/o WikiFT	25	71.9	81.3
w/o WikiFT	25	73.6	82.5
w/o Uℓ_2	25	76.2	84.7

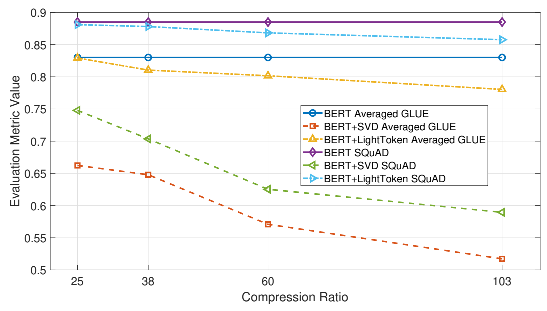


Figure 6: Performance with different compression ratios. The y-axis represents the averaged accuracy of all GLUE tasks for GLUE dataset and represents F1 for SQuAD.

that fine-tuning the decoder is effective to close the gap between the full model and the compressed model.

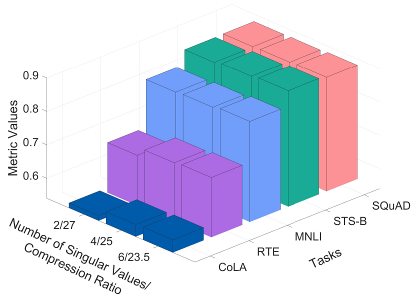


Figure 7: Performance of different k used in rank-k approximation.

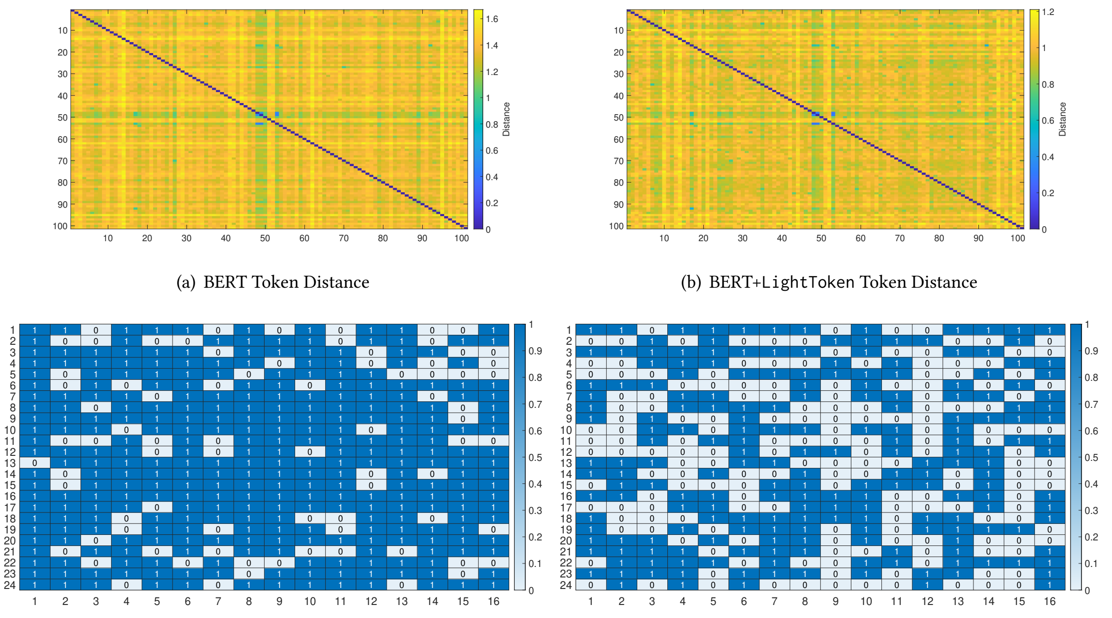
5.3.3 Effectiveness of the Residual Connection. In this section, we do the ablation study to answer RQ2 regarding the residual connection. We conduct this ablation study with BERT and DistilBERT as backbones and show the results in Fig. 5. We find that the residual connection improves model performance on most tasks. For example, it leads to 2.1% and 1.7% improvement on GLUE tasks on average for BERT and DistilBERT respectively. It brings 3.4% improvement on the SQuAD dataset with DistilBERT as the backbone. In particular, the performance of BERT+LightToken w/o the residual, BERT+LightToken, DistilBERT+LightToken w/o residual, and DistilBERT+LightToken are 47.6, 57.6, 46.2, and 51.7 on CoLA dataset, achieving 21.1% and 103.1% improvement compared to the model without the residual connection with BERT and DistilBERT as backbones, respectively. It shows the proposed residual connection is an effective solution to learn more informative hash codes.

5.4 Performance w.r.t. Different Compression Ratios

In this section, we study how the performance of LightToken changes with respect to different compression ratios in order to answer RQ3. We show the average performance on GLUE and F1 value on SQuAD in Fig. 6. We notice that with the increment of the compression ratio, the accuracy performance degradation of the proposed LightToken is very slight. For example, when the compression ratio is 25, LightToken does not have performance drop in term of accuracy both on GLUE and SQuAD. When compression ratio reaches 103, accuracy drop of LightToken is still within 6%. However, for SVD, when the compression ratio is 103, its accuracy drop on GLUE and SQuAD is huge, as large as 37.7% and 33.4% respectively. It shows the superiority of LightToken, especially when we need to highly compress the token embedding matrix.

5.5 Performance w.r.t. Different Ranks of SVD Approximation

In this section, we study how performance changes with respect to different ranks of SVD approximation in order to answer RQ3. We show the performance on datasets CoLA, RTE, MNLI, STS-B, and SQuAD with rank-2, 4, and 6 SVD approximation in Fig. 7. There are two key discoveries. First, model performance increases when using a higher rank approximation. It is not surprising since the reconstruction error will be reduced with more parameters utilized. Second, the rank-4 approximation has better performance



(a) BERT Token Distance

(b) BERT+LightToken Token Distance

(c) dog and dogs

(d) dog and tiger

Figure 8: Fig. 8(a) represents the pairwise distance of original embedding corresponding to sampled tokens. Fig. 8(b) represents the pairwise distance of LightToken reconstructed embedding corresponding to sampled tokens. Fig. 8(c) and Fig. 8(d) visualize the learned hash codes differences. Number 1 means the hash code is the same in the dimension and 0 means the hash code is different in the dimension. The 384 dimension hash codes are reshaped as a 24×16 matrix.

than rank-2 approximation. However, it is observed to bring limited performance improvement when further increasing the rank approximation to rank 6. More precisely, using rank-4 approximation leads to 3.8% and 3.7% relative accuracy performance improvement compared with adopting rank-2 approximation on CoLA and RTE datasets. However, the improvement of rank-6 approximation is less than 0.05% compared to rank-4 approximation. It shows that using the largest four singular values will be sufficient enough to preserve the token information.

5.6 Case Study

In this section, we visualize the reconstructed token representations and hash codes in Fig. 8(a) to answer RQ4. To show that the reconstructed representations can preserve semantics information, we sample 100 token ids from the BERT token vocabulary, and compute the pairwise distance before and after conducting compression in Fig. 8(a) and Fig. 8(b) respectively. The brightness of the color in the two figures indicates the distance. The color patterns in Fig. 8(a) and Fig. 8(b) are very similar, showing the pattern is preserved before and after conducting LightToken compression. If we use a complete graph to model all tokens, where vertices are tokens and edges are weighted by the distances among tokens, the adjacent matrix will be represented by the pairwise distance matrix. Therefore, Fig. 8(a) and Fig. 8(b) illustrate that the reconstructed representations preserve the neighborhood information of original token representations. Besides, to show that the hash codes can maintain semantics information, we select three tokens, including dog, dogs, and tiger, and visualize two of their pairwise representations in Fig. 8(c) and Fig. 8(d). We compute the difference for each bit. For the i-th dimension of hash codes, if their bits are the same, the result for this bit is 1, else is 0. We reshape the 384-dimension hash codes as a 24×16 matrix. Intuitively, the hash codes of dog and dogs should be more similar than that of dog and tiger. Fig. 8(c) and Fig. 8(d) is consistent with the intuition. It demonstrates that the learned hash codes also preserve semantic information.

6 CONCLUSION

In this paper, we proposed a task and model-agnostic lightweight token embedding framework, namely LightToken, for effective compression of PLMs. The proposed LightToken integrates low-rank and hashing approximation. We first leverage SVD to obtain a rank-k approximation matrix. Then the residual between the token embedding matrix and the rank-k approximation matrix is encoded as hash codes by the proposed residual binary autoencoder. The decoder is fine-tuned in the Wikipedia corpus to further reduce accuracy loss. Theoretical analysis demonstrates the benefit of such integration, as shown by the smaller number of hash codes needed after the rank-k approximation. Extensive experiments are conducted on GLUE and SQuAD v1.1 benchmarks. Experimental results show that the proposed LightToken outperforms state-of-the-art baselines, and achieves 25x compression ratio without accuracy loss.

ACKNOWLEDGEMENT

This work is supported in part by the US National Science Foundation under grant NSF IIS-1747614 and IIS-2141037. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Anish Acharya, Rahul Goel, Angeliki Metallinou, and Inderjit Dhillon. 2019. Online embedding compression for text classification using low rank matrix factorization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33. 6196–6203.

[2] Haoli Bai, Wei Zhang, Lu Hou, Lifeng Shang, Jing Jin, Xin Jiang, Qun Liu, Michael Lyu, and Irwin King. 2020. Binarybert: Pushing the limit of bert quantization. *arXiv preprint arXiv:2012.15701* (2020).

[3] Klaudia Bałazy, Mohammadreza Banaei, Rémi Lebret, Jacek Tabor, and Karl Aberer. 2021. Direction is what you need: Improving Word Embedding Compression in Large Language Models. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*. 322–330.

[4] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432* (2013).

[5] Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The Fifth PASCAL Recognizing Textual Entailment Challenge.. In *TAC*.

[6] Patrick Chen, Hsiang-Fu Yu, Inderjit Dhillon, and Cho-Jui Hsieh. 2021. Drone: Data-aware low-rank compression for large nlp models. *Advances in neural information processing systems* 34 (2021), 29321–29334.

[7] Tianlong Chen, Jonathan Frankle, Shiyu Chang, Sijia Liu, Yang Zhang, Zhangyang Wang, and Michael Carbin. 2020. The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems* 33 (2020), 15834–15846.

[8] Ting Chen, Martin Renqiang Min, and Yizhou Sun. 2018. Learning k-way d-dimensional discrete codes for compact embedding representations. In *ICML*. PMLR, 854–863.

[9] Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. 2019. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. IEEE, 3009–3018.

[10] Ido Dagan, Oren Glickman, and Bernardo Magnini. 2006. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*. Springer, 177–190.

[11] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[12] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Herve Jegou, and Armand Joulin. 2020. Training with quantization noise for extreme model compression. *arXiv preprint arXiv:2004.07320* (2020).

[13] Jonathan Frankle and Michael Carbin. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635* (2018).

[14] Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*. 1–9.

[15] R Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second pascal recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, Vol. 7.

[16] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654* (2020).

[17] Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531* 2, 7 (2015).

[18] Yen-Chang Hsu, Ting Hua, Sungen Chang, Qian Lou, Yilin Shen, and Hongxia Jin. 2022. Language model compression with weighted low-rank factorization. *arXiv preprint arXiv:2207.00112* (2022).

[19] Nanjiang Jiang and Marie-Catherine de Marneffe. 2019. Evaluating BERT for natural language inference: A case study on the CommitmentBank. 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)*.

[20] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351* (2019).

[21] Jing Jin, Cai Liang, Tiancheng Wu, Liqin Zou, and Zhiliang Gan. 2021. KDLSQ-BERT: A quantized bert combining knowledge distillation with learned step size quantization. *arXiv preprint arXiv:2101.05938* (2021).

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

[23] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942* (2019).

[24] Dongjun Lee, Sohee Yang, and Minjeong Kim. 2019. CLaF: Open-Source Clova Language Framework. https://github.com/naver/claf.

[25] Vasileios Lioutas, Ahmad Rashid, Krtin Kumar, Md Akmal Haidar, and Mehdi Rezagholizadeh. 2019. Distilled embedding: non-linear embedding factorization using knowledge distillation. (2019).

[26] Shicong Liu, Hongtao Lu, and Junru Shao. 2015. Improved Residual Vector Quantization for High-dimensional Approximate Nearest Neighbor Search. *arXiv preprint arXiv:1509.05195* (2015).

[27] 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. *arXiv preprint arXiv:1907.11692* (2019).

[28] Zejian Liu, Fanrong Li, Gang Li, and Jian Cheng. 2021. EBERT: Efficient BERT Inference with Dynamic Structured Pruning. In *Findings of ACL'21*. 4814–4823.

[29] Yi Ma, Harm Derksen, Wei Hong, and John Wright. 2007. Segmentation of multivariate mixed data via lossy data coding and compression. *IEEE transactions on pattern analysis and machine intelligence* 29, 9 (2007), 1546–1562.

[30] JS McCarley, Rishav Chakravarti, and Avirup Sil. 2019. Structured pruning of a BERT-based question answering model. *arXiv preprint arXiv:1910.06360* (2019).

[31] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models. arXiv:1609.07843 [cs.CL]

[32] Manish Munikar, Sushil Shakya, and Aakash Shrestha. 2019. Fine-grained sentiment classification using BERT. In *2019 Artificial Intelligence for Transforming Business and Society (AITB)*, Vol. 1. IEEE, 1–5.

[33] Antonio Polino, Razvan Pascanu, and Dan Alistarh. 2018. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668* (2018).

[34] Chen Qu, Liu Yang, Minghui Qiu, W Bruce Croft, Yongfeng Zhang, and Mohit Iyyer. 2019. BERT with history answer embedding for conversational question answering. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*. 1133–1136.

[35] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. Squad: 100,000+ questions for machine comprehension of text. *arXiv preprint arXiv:1606.05250* (2016).

[36] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. 2013. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 6655–6659.

[37] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

[38] Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 8815–8821.

[39] Raphael Shu and Hideki Nakayama. 2018. Compressing Word Embeddings via Deep Compositional Code Learning. In *ICLR*.

[40] Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019. Patient knowledge distillation for bert model compression. *arXiv preprint arXiv:1908.09355* (2019).

[41] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984* (2020).

[42] Hanlin Tang, Xipeng Zhang, Kai Liu, Jianchen Zhu, and Zhanhui Kang. 2022. MKQ-BERT: Quantized BERT with 4-bits Weights and Activations. *arXiv preprint arXiv:2203.13483* (2022).

[43] Wei Tang, Gang Hua, and Liang Wang. 2017. How to train a compact binary neural network with high accuracy?. In *AAAI'31*.

[44] Jiayi Tian, Chao Fang, Haonan Wang, and Zhongfeng Wang. 2022. BEBERT: Efficient and robust binary ensemble BERT. *arXiv preprint arXiv:2210.15976* (2022).

[45] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461* (2018).

[46] Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2018. Neural Network Acceptability Judgments. *arXiv preprint arXiv:1805.12471* (2018).

[47] Genta Indra Winata, Andrea Madotto, Jamin Shin, and Elham J Barezi. 2018. Low-Rank Matrix Factorization of LSTM as Effective Model Compression. (2018).

[48] Guangxuan Xiao, Ji Lin, Mickael Seznec, Julien Demouth, and Song Han. 2022. Smoothquant: Accurate and efficient post-training quantization for large language models. *arXiv preprint arXiv:2211.10438* (2022).

[49] Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*. IEEE, 36–39.

[50] Wei Zhang, Lu Hou, Yichun Yin, Lifeng Shang, Xiao Chen, Xin Jiang, and Qun Liu. 2020. Ternarybert: Distillation-aware ultra-low bit bert. *arXiv preprint arXiv:2009.12812* (2020).

[51] Sanqiang Zhao, Raghav Gupta, Yang Song, and Denny Zhou. 2019. Extreme language model compression with optimal subwords and shared projections. (2019).

[52] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. 2016. Trained ternary quantization. *arXiv preprint arXiv:1612.01064* (2016).

[53] Michael Zhu and Suyog Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878* (2017).

A PROOF OF PROPOSITION 1

Lemma 1. *Assume we are given a set of N i.i.d. samples $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_N] \in \mathbb{R}^{d\times N}$ from a multivariate Gaussian distribution. We encode the vector $\boldsymbol{x}$ such that the encoded representation can be used to recover $\boldsymbol{x}$ given a distortion ϵ^2, i.e. $\mathbb{E}[\|\boldsymbol{x}-\hat{\boldsymbol{x}}\|^2] \le \epsilon^2$. Then the average number of bits needed can be represented as*

$$L(X) = \frac{N+d}{2N}\log(\boldsymbol{I} + \frac{d}{\epsilon^2 N}\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^T) + \frac{d}{2N}\log(1+\frac{\boldsymbol{\mu}^T\boldsymbol{\mu}}{\epsilon^2}), \quad (18)$$

where $\boldsymbol{\mu} = \frac{1}{N}\boldsymbol{X}\boldsymbol{1}_{N\times 1}$, and $\bar{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{\mu}\boldsymbol{1}^T_{N\times 1}$. If the $\boldsymbol{X} = [\boldsymbol{x}_1, ..., \boldsymbol{x}_N] \in \mathbb{R}^d$ from a subspace, i.e., a degenerate Gaussian, Eqn. 18 is an upper bound of the code length [29].

Proposition 1. *Assume each row of $\boldsymbol{X}$ is sampled from a multivariate Gaussian distribution and $\frac{1}{N}\boldsymbol{X}\boldsymbol{1}_{N\times 1} = \boldsymbol{0}$. The singular value decomposition of $\boldsymbol{X}$ is $\boldsymbol{X} = \boldsymbol{U}\Lambda\boldsymbol{V}^T$. The k approximate matrix $\boldsymbol{X}_k = \boldsymbol{U}\Lambda_k\boldsymbol{V}^T$, where Λ_k is the same matrix as Λ except that it contains only the k largest singular values. Then we have $L(\boldsymbol{X}-\boldsymbol{X}_k) < L(\boldsymbol{X})$ when $\frac{\Pi_{i=1}^d(1+\frac{d}{N\epsilon^2}\sigma_i^2(\boldsymbol{X}))}{(1+\frac{d}{N\epsilon^2}\sigma_{k+1}^2(\boldsymbol{X}))^d} > (1+\frac{d}{N^2\epsilon^2}\sigma_1^2(\boldsymbol{X}))^{\frac{N}{N+d}}$, where $\sigma_i(\boldsymbol{X})$ is the i-th largest singular value of $\boldsymbol{X}$.*

Proof. According to Lemma 1, we have

$$\begin{aligned} L(\boldsymbol{X}) =& \frac{N+d}{2N}\log(\boldsymbol{I} + \frac{d}{\epsilon^2 N}\boldsymbol{X}\boldsymbol{X}^T) \\ =& \frac{N+d}{2N}\sum_{i=1}^d \log(1+\frac{d}{\epsilon^2 N}\sigma_i^2(\boldsymbol{X})). \end{aligned} \quad (19)$$

Similarly, we have

$$L(\boldsymbol{X}-\boldsymbol{X}_k) \le \frac{N+d}{2N}\log(\boldsymbol{I} + \frac{d}{\epsilon^2 N}\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^T) + \frac{d}{2N}\log(1+\frac{\boldsymbol{\mu}^T\boldsymbol{\mu}}{\epsilon^2}), \quad (20)$$

where $\boldsymbol{\mu} = \frac{1}{N}(\boldsymbol{X}-\boldsymbol{X}_k)\boldsymbol{1}_{N\times 1}$, and $\bar{\boldsymbol{X}} = \boldsymbol{X} - \boldsymbol{X}_k - \boldsymbol{\mu}\boldsymbol{1}^T_{N\times 1}$. Therefore,

$$\begin{aligned} \bar{\boldsymbol{X}} =& \boldsymbol{X} - \boldsymbol{X}_k - \boldsymbol{\mu}\boldsymbol{1}^T_{N\times 1} \\ =& (\boldsymbol{X}-\boldsymbol{X}_k)(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T). \end{aligned} \quad (21)$$

Hence, we have

$$\begin{aligned} \sigma_i(\bar{\boldsymbol{X}}) \le& \sigma_1(\boldsymbol{X}-\boldsymbol{X}_k)\sigma_i(\boldsymbol{I} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T) \\ \le& \sigma_1(\boldsymbol{X}-\boldsymbol{X}_k) \\ =& \sigma_{k+1}(\boldsymbol{X}). \end{aligned} \quad (22)$$

So $\frac{N+d}{2N}\log(\boldsymbol{I} + \frac{d}{\epsilon^2 N}\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^T) \le \frac{N+d}{2N} d\log(1+\frac{d}{\epsilon^2 N}\sigma_{k+1}^2(\boldsymbol{X}))$.
Then consider

$$\begin{aligned} \boldsymbol{\mu}^T\boldsymbol{\mu} =& \frac{1}{N^2}\boldsymbol{1}^T\boldsymbol{X}_k\boldsymbol{X}_k\boldsymbol{1} \\ \le& \frac{1}{N^2}\sigma_1^2(\boldsymbol{X})\boldsymbol{1}^T\boldsymbol{1} \\ =& \frac{1}{N}\sigma_1^2(\boldsymbol{X}), \end{aligned} \quad (23)$$

so we have

$$\frac{d}{2N}\log(1+\frac{\boldsymbol{\mu}^T\boldsymbol{\mu}}{\epsilon^2}) \le \frac{d}{2N}\log(1+\frac{\sigma_1^2(\boldsymbol{X})}{N\epsilon^2}) \quad (24)$$

$$\le \frac{1}{2}\log(1+\frac{d\sigma_1^2(\boldsymbol{X})}{N^2\epsilon^2}). \quad (25)$$

Therefore,

$$\begin{aligned} & L(\boldsymbol{X}) - L(\boldsymbol{X}-\boldsymbol{X}_k) \\ \ge& \frac{N+d}{2N}\sum_{i=1}^d \log(1+\frac{d}{\epsilon^2 N}\sigma_i^2(\boldsymbol{X})) \\ & - \frac{N+d}{2N} d\log(1+\frac{d}{\epsilon^2 N}\sigma_{k+1}^2(\boldsymbol{X})) - \frac{1}{2}\log(1+\frac{d\sigma_1^2(\boldsymbol{X})}{N^2\epsilon^2}) \\ >& 0. \end{aligned} \quad (26)$$

□

B ALGORITHM OF LIGHTTOKEN

Algorithm 1: LightToken

```
Input: A pre-trained language model; Wikipedia corpus $\mathfrak{X}$;
    downstream task dataset $\mathfrak{X}_t$; rank of SVD approxiamtion $k$.
// first stage: rank-$k$ SVD approximation
1  $[\boldsymbol{U}, \boldsymbol{S}, \boldsymbol{V}] = \mathrm{SVD}(\boldsymbol{X})$;
2  if PLM is RoBERTa then
3      $diag(\boldsymbol{S})[k+1:] = diag(\boldsymbol{S})[k+1:]/2$;
4  Compute SVD approximation matrix $\boldsymbol{U}_{X,k}\boldsymbol{V}^T_{X,k}$;
// second stage: learning hash codes
5  Compute the residual between original token embedding
     matrix $\boldsymbol{x}$ and SVD approximation matrix $\boldsymbol{U}_{X,k}\boldsymbol{V}^T_{X,k}$:
     $\boldsymbol{R} = \boldsymbol{X} - \boldsymbol{U}_{X,k}\boldsymbol{V}^T_{X,k}$;
6  while converge do
7      Compute reconstructed token via residual binary
         autoencoder;
8      Compute U$\ell_2$ loss based on Eqn. 13 and do
         backpropagation to update parameters;
// third stage: fine-tune decoder
9  Replace the token embedding matrix of PLM with
     $\boldsymbol{U}_{X,k}\boldsymbol{V}^T_{X,k} + \mathrm{Dec}(\boldsymbol{B})$;
10 Fix parameters of the PLM except token embedding $\Theta_{PLM}$
     and hash codes $\boldsymbol{B}$;
11 while converge do
12     for batch in $\mathfrak{X}$ do
13         Forward propagation based on Eqn. 15;
14         Update $\Theta_D$;
// fourth stage: fine-tune PLM for downstream
    task
15 Fix $\boldsymbol{B}$;
16 while converge do
17     for batch in $\mathfrak{X}_t$ do
18         Forward propagation based on Eqn. 16;
19         Update $\Theta_D$ and $\Theta_{PLM}$;
```

C DATASET AND IMPLEMENTATION DETAILS

We use rank-4 SVD approximation and two residual layers for all backbones. For `LightToken` with BERT and DistilBERT as backbones, the decoder is a one-hidden-layer MLP with ReLU as the activation function. For `LightToken` with RoBERTa as the backbone, the decoder is a two-hidden-layers MLP with GeLU as the activation function. We use wikitext-103-raw-v1 [31] as the corpus to fine-tune the decoder. The decoder is fine-tuned for 10 epochs. We select the learning rates from $\{5e-6, 1e-5, 2e-5, 3e-5, 5e-5\}$ and batch size from {8, 16, 32, 64}. Experiments are conducted using four NIVIDA RTX A6000.

Table 3: Detail information of GLUE and SQuAD v1.1.

Dataset	Task	Metric
MRPC	Paraphrase Identification	F1
QQP	Paraphrase Identification	F1
MNLI	Natural Language Inference	Accuracy
QNLI	Natural Language Inference	Accuracy
RTE	Natural Language Inference	Accuracy
SST-2	Sentiment Classification	Accuracy
STS-B	Similarity	Pearson corr
CoLA	Linguistic Acceptability	Matthews corr
SQuAD v1.1	Question Answering	Exact Match/F1