BBTv2: Pure Black-Box Optimization Can Be Comparable to Gradient Descent for Few-Shot Learning

Tianxiang Sun
Fudan University
txsun19@fudan.edu.cn
Zhengfu He
Fudan University
zfhe19@fudan.edu.cn
Hong Qian
East China Normal University
hqian@cs.ecnu.edu.cn
Xuanjing Huang
Fudan University
xjhuang@fudan.edu.cn
Xipeng Qiu
Fudan University
xpqiu@fudan.edu.cn

Abstract
Black-Box Tuning (BBT) is a derivative-free approach to optimize continuous prompt tokens prepended to the input of language models. Although BBT has achieved comparable performance to full model tuning on simple classification tasks under few-shot settings, it requires pre-trained prompt embedding to match model tuning on hard tasks (e.g., entailment tasks), and therefore does not completely get rid of the dependence on gradients. In this paper we present BBTv2, a pure black-box optimization approach that can drive language models to achieve comparable results to gradient-based optimization. In particular, we prepend continuous prompt tokens to every layer of the language model and propose a divide-and-conquer algorithm to alternately optimize the prompt tokens at different layers. For the optimization at each layer, we perform derivative-free optimization in a low-dimensional subspace, which is then randomly projected to the original prompt parameter space. Experimental results show that BBTv2 not only outperforms BBT by a large margin, but also achieves comparable or even better performance than full model tuning and state-of-the-art parameter-efficient methods (e.g., Adapter, LoRA, BitFit, etc.) under few-shot learning settings, while maintaining much fewer tunable parameters.\(^1\)

1 Introduction
Pre-Trained Models (PTMs) (Devlin et al., 2019; Liu et al., 2019; Yang et al., 2019; Lewis et al., 2020; Raffel et al., 2020; Qiu et al., 2020) have pushed the state-of-the-art of many NLP tasks. Especially, supersized PTMs such as GPT-3 (Brown et al., 2020) can easily adapt to downstream tasks even with a few labeled samples. However, it is expensive for most users to locally run such supersized models. In addition, parameters of some large-scale PTMs are inaccessible due to commercial reasons. Hence, supersized PTMs are often released as public services, allowing users to access the models through black-box APIs.

In such a scenario, namely Language-Model-as-a-Service (LMaaS) (Sun et al., 2022b), considerable performance has been observed in many use cases (Brown et al., 2020). For instance, one can hand-craft textual prompts and include some labeled samples in the prompts (a.k.a. in-context learning (Brown et al., 2020)) that are then concatenated with input texts to query PTMs. Nevertheless, in-context learning heavily relies on human-designed prompts and suffers from high variances. Recently, Sun et al. (2022b) propose the black-box tuning (BBT), which optimizes continuous task-specific prompts using derivative-free optimization (DFO) algorithms. Although it has been demonstrated that, BBT with RoBERTa (Liu et al., 2019) achieves comparable or better performance than full model tuning on simple text classification tasks

\(^1\)Work in progress. Code is publicly available at https://github.com/txsun1997/Black-Box-Tuning.
(e.g., sentiment analysis), it lacks versatility across tasks and language models. BBT struggles on more complicated tasks: on hard tasks (e.g., natural language inference), BBT still under-performs model tuning if not using pre-trained prompt embedding; the convergence of BBT can be slow when the number of classes becomes large (e.g., fine-grained topic classification tasks). Besides, our pilot experiments ($\S2.2$) show that, when switching to other PTMs, BBT has no promise in generalizing to unseen data, though it fits well on training data.

In this paper, we present BBTv2 to improve BBT across tasks and language models. We draw inspiration from two lines of work: (1) deep prompt tuning and (2) divide-and-conquer (DC) methods for high-dimensional DFO. On the one hand, deep prompt tuning, which is to prepend and optimize continuous prompt tokens at each layer of the PTM, has been demonstrated to significantly improve performance on various language understanding and generation tasks (Li and Liang, 2021; Qin and Eisner, 2021; Liu et al., 2021b). Hence, a straightforward extension of BBT is to optimize the deep prompts instead of the prompts merely in the input layer. However, the deep prompts contain an order of magnitude more parameters, posing a challenge for high-dimensional DFO. On the other hand, fortunately, we find that the forward computation of modern PTMs can be decomposed into an additive form w.r.t. the hidden states of each layer thanks to the residual connections. Therefore, the high-dimensional optimization problem can be decomposed into multiple low-dimensional subproblems, each corresponding to the prompts at one layer. Based on this insight, we propose a divide-and-conquer algorithm to alternately optimize the prompt tokens at each layer. For the optimization at each layer, we maintain a random projection that maps the prompt parameters into a lower-dimensional subspace and perform derivative-free optimization in the generated subspace.

Experimental results show that BBTv2 significantly improves BBT on average performance across 7 language understanding tasks. As shown in Figure 1, BBTv2 also achieves comparable or even better performance than full model tuning and state-of-the-art parameter-efficient tuning methods including Adapter (Houlsby et al., 2019), BitFit (Zaken et al., 2022), LoRA (Hu et al., 2021), and P-Tuning v2 (Liu et al., 2021b), while with much fewer tunable parameters.

**2 Pilot Experiments**

First we conduct pilot experiments to demonstrate the limitations of BBT across tasks and models.

**2.1 Limitations Across Tasks**

**Unsatisfactory Performance on Entailment Tasks.** As demonstrated by Sun et al. (2022b), BBT can outperform model tuning on entailment tasks when using pre-trained prompt embedding for initialization. However, pre-trained prompt embedding is not always available for many languages and models. Without pre-trained prompt embedding, BBT is still lagging behind model tuning on entailment tasks. In other words, BBT does not completely get rid of the dependence on gradients to exceed model tuning. In contrast, as depicted in Figure 2, the proposed BBTv2 can match or outperform full model tuning on entailment tasks, namely MRPC (Dolan and Brockett, 2005), SNLI (Bowman et al., 2015), and RTE (Wang et al., 2019) without using pre-trained prompt embedding for initialization.

**Slow Convergence on Classification Tasks with Many Classes.** BBT suffers from slow convergence rate when the number of classes becomes large. As reported by Sun et al. (2022b), BBT cannot converge within a budget of 8,000 API calls, which is sufficient for common tasks to converge, on DBPedia (Zhang et al., 2015), a topic classification task with 14 classes. Figure 3 shows the cross entropy loss and the accuracy on the training set during training. Compared with BBT, the proposed BBTv2 significantly accelerate the convergence on DBPedia.
2.2 Limitations Across Models

Overfitting on Training Data. When switching the backbone model from RoBERTa (Liu et al., 2019) to other PTMs, we find that BBT tends to overfit training data. As shown in Figure 4, the original BBT with BERT\textsubscript{LARGE} (Devlin et al., 2019) and BART\textsubscript{LARGE} (Lewis et al., 2020) can achieve 100% accuracy on the SST-2 training set, but achieves little improvement on the development set. We conjecture that the random projection adopted by the original BBT hinders its generalization. By generating random projections with normal distributions (§3.3), our modified BBT and BBTv2 exhibit stronger generalization ability.

3 Methods

We first introduce the original BBT in §3.1, and then describe the proposed BBTv2 in §3.2. In §3.3, we revisit the random projection with normal distribution.

3.1 Black-Box Tuning

Black-Box Tuning (BBT) (Sun et al., 2022b) is a derivative-free framework to drive PTMs for few-shot learning. In particular, for a batched training data \((X, Y)\), we first convert the texts \(X\) with some pre-defined templates (e.g., "It was [MASK]") into \(\tilde{X}\), and the labels \(Y\) with a pre-defined map into label words \(\tilde{Y}\) (e.g., "great" and "terrible"). By this, we can formulate various downstream tasks into a general-purpose (masked) language modeling task and utilize the pre-trained (masked) language modeling head to solve them. Assume the PTM inference API \(f\) takes a continuous prompt \(p\) and a batch of texts \(\tilde{X}\) as input, and outputs the logits of the tokens of interest (e.g., the [MASK] token). BBT seeks to find the optimal prompt \(p^* = \arg\min_{p \in \mathcal{P}} \mathcal{L}(f(p; \tilde{X}), \tilde{Y})\), where \(\mathcal{P}\) is the prompt space and \(\mathcal{L}\) is some loss function such as cross entropy. The closed form and the gradients of \(f\) are not accessible to BBT.

Since the prompt \(p \in \mathbb{R}^D\) usually has tens of thousands of dimensions, making it infeasible to be optimized with derivative-free optimization (DFO) algorithms. Hence, BBT adopts a random projection \(A \in \mathbb{R}^{D \times d}\) to generate a low-dimensional subspace \(Z \in \mathbb{R}^d\) and performs optimization in the generated subspace, i.e.,

\[
z^* = \arg\min_{z \in Z} \mathcal{L}(f(Az + p_0; \tilde{X}), \tilde{Y}),
\]

where \(p_0\) is the initial prompt embedding. If not using pre-trained prompt embedding, \(p_0\) is the word embeddings randomly sampled from the PTM vocabulary.
BBT adopts the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (Hansen and Ostermeier, 2001; Hansen et al., 2003) to optimize Eq.(1) and obtain the desired prompt \( p^* = A z^* \). The random projection matrix \( A \) is frozen during optimization.

### 3.2 BBTv2: Deep Black-Box Tuning

Though BBT achieved comparable performance to model tuning on simple classification tasks, our pilot experiments (§2) show that it lacks versatility across tasks and language models. As an improved variant of BBT, BBTv2 seeks to generalize BBT across tasks and models through deep prompts.

Inspired by the success of deep prompt tuning (Li and Liang, 2021; Qin and Eisner, 2021; Liu et al., 2021b), we manage to inject continuous prompt tokens to every layer of the PTM and optimize them with DFO methods. Compared to BBT that optimizes the prompts merely in the input layer, BBTv2 has an order of magnitude more parameters. For a PTM with \( L \) layers, BBTv2 seeks to optimize \( p = \langle p_1, \ldots, p_L \rangle \), where \( p_i \in \mathbb{R}^D \). Hence, the number of parameters to be optimized becomes \( LD \). Say we are using RoBERTa\_LARGE with 24 layers and insert 50 prompt tokens at each layer, the total number of parameters to be optimized is 1.2M. Instead of simply extending the dimension of the random projection matrix \( A \) to \( LD \times d \), we propose a divide-and-conquer (DC) algorithm to handle the increased parameters.

In fact, DC is another effective technique to cope with high-dimensional DFO problems by decomposing the original high-dimensional problem into multiple low-dimensional subproblems, and solving them separately (Kandasamy et al., 2015; Mei et al., 2016). The key assumption of applying DC is that, the objective \( f \) can be decomposed into some additive form. Fortunately, modern PTMs can be expanded into an additive form due to the residual connections (He et al., 2016). For instance, a three-layered PTM can be decomposed as

\[
\begin{align*}
    f(x_1) &= f_3(x_3) + x_3 & (2) \\
    &= f_3(x_3) + f_2(x_2) + x_2 & (3) \\
    &= f_3(x_3) + f_2(x_2) + f_1(x_1) + x_1, & (4)
\end{align*}
\]

where \( f_i \) is the transformation function of the \( i \)-th continuous prompts \( x_j \), and the \( x_1 \) is the input of the \( i \)-th layer, and \( x_1 \) is the input embedding. Thus, optimizing the continuous prompts \( \{ p_j \}_{j=1}^L \) attached to the hidden states at every layer \( \{ x_j \}_{j=1}^L \), can be regarded as independent subproblems.\(^2\) Since the assumption is satisfied, we propose a DC-based algorithm, which is described in Algorithm 1, to implement BBTv2.

The prompts at different layers are optimized alternately from bottom to up. For the optimization at each layer, we maintain a specific random projection \( A_j \) and a CMA-ES optimizer \( M_j \). When alternating to layer \( j \) (Line 5-6 in Algorithm 1), a single CMA-ES iteration is performed in the same fashion as BBT, i.e., a new \( z_j \) is generated by \( M_j \) and is then projected to \( p_j \) using \( A_j \).

During PTM inference, \( p_j = A_j z_j \) is first added with an initial prompt embedding \( p_0^j \) and then concatenated with the hidden states \( x_j \). Thus, according to Eq.(4), the computation of a \( L \)-layered PTM can be viewed as

\[
\begin{align*}
f(x_1; p) &= [A_1 z_1 + p_0^1; x_1] \\
&+ \sum_{j=1}^{L} f_j([A_j z_j + p_0^j; x_j]), & (5)
\end{align*}
\]

\(^2\) We omit the classification head on the top of the PTM since it is usually a linear transformation and would not change the additive decomposition.

---

**Algorithm 1: Black-Box Tuning v2**

**Require:** \( L \)-layer PTM Inference API \( f \); Budget of API calls \( B \); Derivative-free optimizers \( \{ M_j \}_{j=1}^L \)

1. Initialize random projections \( A_1, \ldots, A_L \) and parameters to be optimized \( z_1^{(0)}, \ldots, z_L^{(0)} \)
2. Calculate initial prompts \( p = \langle A_1 z_1^{(0)}, \ldots, A_L z_L^{(0)} \rangle \)
3. for \( i = 1 \) to \( B / L \)
   4. for \( j = 1 \) to \( L \)
      5. \( z_j^{(i)} \leftarrow M_j(f, \langle p_1, \ldots, p_{j-1}, A_j z_j^{(i-1)}, p_{j+1}, \ldots, p_L \rangle) \)
      6. \( p_j \leftarrow A_j z_j^{(i)} \)
   7. end for
5. end for
6. return Optimized deep prompts \( p = \langle p_1, \ldots, p_L \rangle \)
where \([\cdot; \cdot]\) means concatenation. Tunable parameters are highlighted in color. Following Sun et al. (2022b), we set \(p_0^1\) as the word embeddings randomly drawn from the PTM vocabulary. \(p_j^0 (1 < j < L)\) is the hidden states of the prompt tokens at the \(j\)-th layer.

### 3.3 Revisiting Random Projections

In the original BBT, each entry in the random projection \(A\) is sampled from a uniform distribution (He et al., 2015). In their experiments, using normal distribution \(\mathcal{N}(0, 1/d)\) to generate the random projection results in slow convergence and inferior performance. However, we show in pilot experiments that the uniform distribution exhibits poor generalization when using other PTMs than RoBERTa. In this section, we shed some light on the effect of the random projection, and propose to use normal distributions with model-related means and standard deviations to generate random projections. In fact, most prior works in high-dimensional DFO (Wang et al., 2016; Qian et al., 2016; Letham et al., 2020) also adopt normal distributions to generate random projections. However, they usually simply use \(\mathcal{N}(0, 1/d)\) or \(\mathcal{N}(0, 1)\).

To take a closer look into the effect of the random projection, we draw distribution of the initial prompts \(p\) that are projected from \(z\) by the projection matrix \(A\). Here, \(z\) is sampled from the normal distribution maintained by the CMA-ES, which is initially set to \(\mathcal{N}(0, 0.5)\) in BBT. By generating \(A\) from different distributions, we obtain the distribution of the projected prompts and compare with the distribution of RoBERTa\_LARGE embedding.

As revealed by Figure 5, when \(A\) is sampled from the normal distribution used in the original BBT, the projected prompts \(p\) cannot cover the range of word embeddings, and therefore suffers from slow convergence. In contrast, using uniform distribution can cover the range of embeddings, which explains why it performs well on RoBERTa\_LARGE.

Thus, to generalize BBT (and BBTv2) across different language models, we have to take into account the distribution of word embeddings (and hidden states for BBTv2) of the PTM for generating random projections. In particular, we use the normal distribution with mean and standard deviation as follows,

\[
\mu = \frac{\hat{\mu}}{d - \hat{\sigma}^2},
\]

\[
\sigma = \frac{\hat{\sigma}}{\sqrt{d - \hat{\sigma}^2}}.
\]

where \(\hat{\mu}\) and \(\hat{\sigma}\) are observed mean and standard deviation of word embeddings (and hidden states for BBTv2). The main idea behind the above calculations is to match the distribution between projected prompts and word embeddings (and hidden states for BBTv2). Detailed derivation of Eq.(6) and Eq.(7) can be found in Appendix A. As simulated in Figure 5, when using our designed normal distribution to generate the random projection, the distribution of the projected prompts well matches the shape of word embeddings, and therefore leads to faster convergence and stronger generalization.

### 4 Experiments

#### 4.1 Datasets and Tasks

For comparison, we mainly evaluate on the same datasets as BBT, i.e., SST-2 (Socher et al., 2013), Yelp polarity (Zhang et al., 2015), AG’s News (Zhang et al., 2015), DBPedia (Zhang et al., 2015), SNLI (Bowman et al., 2015), RTE (Wang et al., 2019), and MRPC (Dolan and Brockett, 2005). SST-2 and Yelp are sentiment analysis tasks, AG’s News and DBPedia are topic classification tasks, SNLI and RTE are natural language inference (NLI) tasks, and MRPC is a paraphrase task. In addition, we include two Chinese tasks,
Table 1: Overall comparison on various language understanding tasks. We report mean and standard deviation of performance over 3 different splits (§4.1). All of the results are obtained with pre-trained RoBERTa LARGEx

| Method             | Tunable | SST-2 acc | YELP P. acc | AG’s News acc | DBPedia acc | MRPC F1 | SNLI acc | RTE acc | Avg. |
|--------------------|---------|-----------|-------------|---------------|-------------|---------|---------|---------|------|
| Model Tuning       | 355M    | 85.39 ± 0.84 | 91.82 ± 0.79 | 86.36 ± 1.85 | 97.98 ± 0.14 | 73.35 ± 5.70 | 54.64 ± 7.59 | 58.60 ± 6.21 | 78.88 |
| Adapter            | 2.4M    | 83.91 ± 0.90 | 90.99 ± 2.56 | 86.01 ± 2.18 | 97.99 ± 0.07 | 69.20 ± 3.58 | 51.46 ± 6.63 | 48.62 ± 4.74 | 76.31 |
| BitFit             | 172K    | 81.19 ± 6.08 | 88.63 ± 6.60 | 86.83 ± 0.62 | 94.42 ± 0.94 | 66.26 ± 6.81 | 53.42 ± 10.63 | 52.59 ± 7.31 | 74.76 |
| LoRA               | 786K    | 88.49 ± 2.90 | 90.21 ± 4.00 | 87.09 ± 0.85 | 97.86 ± 0.17 | 72.24 ± 2.23 | 61.03 ± 8.55 | 49.22 ± 15.12 | 78.01 |
| Prompt Tuning      | 50K     | 68.23 ± 3.78 | 61.02 ± 0.65 | 84.61 ± 0.66 | 67.75 ± 1.48 | 51.61 ± 3.67 | 36.13 ± 1.51 | 54.69 ± 3.79 | 63.46 |
| P-Tuning v2        | 1.2M    | 64.33 ± 3.05 | 92.63 ± 1.39 | 83.46 ± 1.00 | 97.05 ± 0.41 | 68.14 ± 3.89 | 36.89 ± 0.79 | 50.78 ± 2.28 | 70.47 |

| Gradient-Free Methods                                      |
|-------------------------------------------------------------|
| Manual Prompt                                               |
| 0                                                           | 79.82 | 89.65 | 76.96 | 41.33 | 67.40 | 31.11 | 51.62 | 62.56 |
| In-Context Learning                                         |
| 0                                                           | 79.79 | ± 0.16 | 85.38 | ± 0.92 | 62.01 | ± 13.34 | 34.83 | ± 7.59 | 45.81 | ± 6.67 | 47.11 | ± 0.63 | 60.36 | ± 1.56 | 59.36 |
| Feature-MLP                                                 |
| 1M                                                          | 64.80 | ± 1.78 | 79.20 | ± 2.26 | 70.77 | ± 0.67 | 87.78 | ± 0.61 | 68.40 | ± 0.86 | 42.01 | ± 0.33 | 53.43 | ± 0.57 | 66.63 |
| Feature-BiLSTM                                              |
| 17M                                                         | 65.95 | ± 0.99 | 74.68 | ± 0.10 | 77.28 | ± 2.83 | 90.37 | ± 3.10 | 71.55 | ± 7.10 | 46.02 | ± 30.38 | 52.17 | ± 2.25 | 68.29 |
| BBT                                                         | 500   | 89.56 | ± 0.25 | 91.50 | ± 0.16 | 81.51 | ± 0.79 | 79.99 | ± 2.99 | 61.56 | ± 4.34 | 46.58 | ± 1.33 | 52.59 | ± 2.21 | 71.90 |
| BB Tv2                                                      | 12K   | 90.98 | ± 0.75 | 90.83 | ± 0.47 | 84.12 | ± 1.40 | 91.79 | ± 0.73 | 76.87 | ± 1.58 | 60.62 | ± 1.01 | 61.73 | ± 2.90 | 79.56 |

4.2 Baselines

We consider two types of methods as our baselines: gradient-based methods and gradient-free methods.

For gradient-based methods, we compare with (1) Model Tuning and state-of-the-art parameter-efficient methods including (2) Adapter (Houlsby et al., 2019), (3) BitFit (Zaken et al., 2022), (4) LoRA (Hu et al., 2021), (5) Prompt Tuning (Lester et al., 2021), and (6) P-Tuning v2 (Liu et al., 2021b). We implement Adapter, BitFit, and LoRA using OpenDelta\(^3\), and evaluate P-Tuning v2 in our experimental settings based on the official implementation\(^6\). The results of Model Tuning and Prompt Tuning are taken from Sun et al. (2022b).

For gradient-free methods, we compare with two non-learning prompt-based methods: (1) Manual Prompt and (2) In-Context Learning (Brown et al., 2020), two feature-based methods: (3) Feature-MLP and (4) Feature-BiLSTM, which is to train a MLP/BiLSTM classifier on the features extracted by the PTM; and (5) BBT (Sun et al., 2022b). The results of these gradient-free baselines are taken from Sun et al. (2022b). One exception is the performance of BBT on DBPedia: in the original paper, BBT is performed given a larger budget (20,000 API calls) on DBPedia for convergence. In this work, we reimplement BBT on DBPedia with the same budget (8,000 API calls) as the other tasks for fair comparison.

4.3 Implementation Details

**Backbones.** To compare with BBT, we mainly use RoBERTaLARGE (Liu et al., 2019) as our backbone model. To verify the versatility of BBTv2, we also evaluate on another discriminative PTM, BERTLARGE (Devlin et al., 2019), and a generative PTM, BARTLARGE (Lewis et al., 2020). In addition, we also evaluate BBTv2 on a supersized Chinese generative PTM, CPM-2 (Zhang et al., 2021b), which has ∼11B parameters.

**Hyperparameters.** Most of the hyperparameters remain the same as BBT: we insert 50 continuous prompt tokens at each layer; the subspace dimen-

\(^3\)https://github.com/SophonPlus/ChineseNlpCorpus
\(^6\)https://github.com/THUDM/P-tuning-v2
4.4 Results

Overall Comparison. As shown in Table 1, BBTv2 outperforms BBT and other gradient-free methods on 6/7 tasks. In contrast to BBT, the improvement of BBTv2 mainly comes from DBPedia, which has 14 classes, and hard entailment tasks, namely MRPC, SNLI, and RTE. On simple tasks such as SST-2 and Yelp, BBT can perform on par with BBTv2. When compared with gradient-based methods, BBTv2 still achieves the best results on average across the 7 tasks while maintaining much fewer tunable parameters. It is worth noting that BBTv2, without any gradient-based components (e.g., the pre-trained prompt embedding used in BBT on entailment tasks (Sun et al., 2022b) or the white-box prompt optimization required by Diao et al. (2022)), is the first purely black-box method that matches the performance of gradient-based optimization on various understanding tasks.

Detailed Comparison. In Table 2, we compare BBTv2 with BBT in other dimensions than accuracy. In addition to the improvement in accuracy, BBTv2 also confers faster convergence than BBT in most cases. For fair comparison of training time, we perform early stopping for both BBT and BBTv2.

BBTv2 Across Models. To verify the universality of BBTv2 across language models, we also evaluate on BERTLARGE and BARTLARGE. As shown in Table 3, BBTv2 can achieve considerable performance across different types of PTMs, i.e., discriminative and generative PTMs. In addition, we also verify the effectiveness of BBTv2 on a supersized Chinese PTM, CPM-2, which has \( \sim 11B \) parameters. As shown in Table 4, when using CPM-2 as the backbone, BBTv2 outperforms model tuning on the two Chinese tasks. The results of Vanilla PT, Hybrid PT, and LM Adaption, which are three variants of prompt tuning, are taken from Gu et al. (2021). We run CPM-2 on 4 NVIDIA A100 GPUs.

4.5 Ablations

Effect of Subspace Dimensionality. The \( d \)-dimensional subspace is the space where the optimization actually performs. We explore the subspace dimensionality from 10 to 1000 using both BBT and BBTv2. The population size is set to \( \lambda = 4 + 3 \log(d) \) accordingly. For each subspace

---

### Table 2: Test accuracy, training time, memory footprint, and the amount of data to be uploaded/downloaded of BBT and BBTv2.

| Method        | SST-2 | AG’s News | DBPedia |
|---------------|-------|-----------|---------|
| BBT           | 89.4  | 82.6      | 89.53   |
| BBTv2         | 91.4  | 85.5      | 87.10   |
| Training Time |       |           |         |
| PyTorch (mins)| 14.8  | 21.0      | 6.1     |
| ONNX (mins)   | 6.1   | 17.7      | 4.6     |
| Memory        |       |           |         |
| User (MB)     | 30    | 30        | 3.0     |
| Server (GB)   | 143   | 143       | 4.6     |
| Network       |       |           |         |
| Upload (KB)   | 6     | 22        | 0.25    |
| Download (KB)| 0.25  | 1         | 1       |

### Table 3: Results of BBTv2 on RoBERTa, BERT, and BART. We use the large version of the three PTMs.

| Method     | SST-2 | AG’s News | DBPedia |
|------------|-------|-----------|---------|
| RoBERTa    | 90.98 ±0.75 | 84.12 ±1.40 | 91.79 ±0.73 |
| BERT       | 77.48 ±2.93 | 78.50 ±0.99 | 93.74 ±0.50 |
| BART       | 89.53 ±2.02 | 81.30 ±2.58 | 87.10 ±2.01 |

### Table 4: Results on two Chinese tasks with CPM-2 as the backbone PTM.

| Method       | Tunable. Params | ChnSent acc | LCQMC acc |
|--------------|-----------------|-------------|-----------|
| Model Tuning | 11B              | 86.1 ±1.8   | 58.8 ±1.8 |
| Vanilla PT   | 410K             | 62.1 ±3.1   | 51.5 ±3.4 |
| Hybrid PT    | 410K             | 79.2 ±4.0   | 54.6 ±2.3 |
| LM Adaption  | 410K             | 74.3 ±5.2   | 51.4 ±2.9 |
| BBTv2        | 4.8K             | 86.4 ±0.8   | 59.1 ±2.5 |

---

7https://onnxruntime.ai/
Cross Entropy Loss

Test Accuracy

| d   | Test Accuracy |
|-----|---------------|
| 30  | 0.00          |
| 35  | 0.35          |
| 40  | 0.40          |
| 45  | 0.45          |
| 50  | 0.50          |
| 55  | 0.55          |
| 60  | 0.60          |
| 65  | 0.65          |
| 70  | 0.70          |

Figure 6: Ablation results on subspace dimensionality and prompt length. We show mean and standard deviation of performance over 5 runs with different random seeds.

Figure 7: Ablation results on the random projection with different distributions. When using our designed normal distribution to generate random projections, both BBT and BBTv2 achieve fast and stable convergence.

dimensionality, we perform 5 runs with different random seeds and record the mean and standard deviation. Experimental results on SST-2 and SNLI are demonstrated in Figure 6, from which we find that: (1) Increasing subspace dimensionality \( d \) generally confers improved performance for both BBT and BBTv2, but marginal effect is also observed when \( d > 100 \); (2) BBTv2 almost always performs better than BBT with the same subspace dimensionality.

Effect of Prompt Length. As reported in prior work (Lester et al., 2021; Sun et al., 2022b), prompt length can be a sensitive hyperparameter to affect the model performance. In our context, prompt length determines the dimensionality of the original prompt parameter space. Hence, we explore the prompt length from 5 to 100 using BBT and BBTv2. Also, we perform 5 runs for each prompt length and report the mean and standard deviation. As shown in Figure 6: (1) The optimal prompt length lies in the range from 5 to 100 and varies across tasks; (2) The effect of prompt length is somehow consistent between BBT and BBTv2.

On Convergence of Normal Distributions. Previously in Figure 4, we show that using our designed normal distribution leads to better generalization from training data to development data. Nevertheless, as reported by Sun et al. (2022b), using normal distributions can suffer from slow convergence. Therefore, we compare the convergence rates using random projections generated from different distributions. As demonstrated in Figure 7: (1) For BBT, the convergence rate of using our designed normal distribution is significantly faster than the normal distribution used in the original BBT, and is comparable to uniform distribution; (2) For BBTv2, using our normal distribution converges more stably on both SST-2 and AG’s News. Especially, we observe that using our normal distribution converges faster than uniform distribution on AG’s News.

5 Related Work

Parameter-Efficient Tuning. In order to reduce the computation and storage cost of large PTMs, much effort has been devoted to parameter-efficient tuning (PET), which is to optimize only a small portion of parameters while keeping the main body of the model unchanged. Thanks to the low intrinsic dimensionality of PTMs (Aghajanyan et al., 2021), PET can achieve comparable performance to full model tuning when training data is sufficient (He et al., 2021). The tunable parameters can be incorporated into different positions of the PTM. Houlsby et al. (2019) insert lightweight neural adapters to each layer of the PTM; Lester et al. (2021) prepend continuous prompt tokens to the input layer; Li and Liang (2021); Liu et al. (2021b) inject tunable prompt tokens to hidden states of
every layer; Zaken et al. (2022) only optimize the bias-terms in the PTM; Hu et al. (2021) learn to adapt attention weights via low-rank matrices. Though the number of tunable parameters is reduced, back-propagation through the entire model is still required to calculate the gradients to update the small portion of parameters. Hence, BBT (Sun et al., 2022b) is proposed to optimize the continuous prompt tokens prepended to the input layer with derivative-free optimization (DFO). This work improves BBT with deep prompts that are injected to every layer of the PTM.

**Prompt-Based Learning.** Another line of work that is related to this work is prompt-based learning, which is to formulate downstream tasks as a (masked) language modeling task, and therefore reduces the gap between PTM pre-training and fine-tuning (Liu et al., 2021a; Sun et al., 2022a). The prompt can be manually designed (Brown et al., 2020; Schick et al., 2020; Schick and Schütze, 2021), mined from corpora (Jiang et al., 2020), generated by paraphrasing (Jiang et al., 2020) or generative PTMs (Gao et al., 2021), or be constructed using gradient-guided search (Shin et al., 2020). In this work, we also insert manually crafted textual prompts into input samples but only optimize the continuous prompt tokens prepended to the hidden states of each layer.

### 6 Conclusion

In this work, we present BBTv2, an improved variant of BBT (Sun et al., 2022b) with deep prompts that are attached to every layer of the PTM. To optimize the high-dimensional prompt parameters, we propose a divide-and-conquer (DC) algorithm combined with random projections to alternately optimize the continuous prompt tokens at each layer. Besides, by simulating the distribution of the generated prompts, we shed some light on the effect of random projections, and propose to use normal distributions with model-related means and standard deviations to generate random projections. Experimental results demonstrate that the proposed BBTv2, without any gradient-based component, can achieve comparable or even better performance than state-of-the-art parameter-efficient tuning methods and full model tuning while maintaining much fewer tunable parameters.

### References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. **Intrinsic dimensionality explains the effectiveness of language model fine-tuning.** In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 7319–7328. Association for Computational Linguistics.

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. **A large annotated corpus for learning natural language inference.** In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP 2015, Lisbon, Portugal, September 17-21, 2015*, pages 632–642. The Association for Computational Linguistics.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. **Language models are few-shot learners.** In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020*, virtual.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. **BERT: pre-training of deep bidirectional transformers for language understanding.** In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Shizhe Diao, Xuechun Li, Yong Lin, Zhichao Huang, and Tong Zhang. 2022. **Black-box prompt learning for pre-trained language models.** *CoRR*, abs/2201.08531.

William B. Dolan and Chris Brockett. 2005. **Automatically constructing a corpus of sentential paraphrases.** In *Proceedings of the Third International Workshop on Paraphrasing, IJCNLP 2005, Jeju Island, Korea, October 2005*, 2005. Asian Federation of Natural Language Processing.

Tianyu Gao, Adam Fisch, and Danqi Chen. 2021. **Making pre-trained language models better few-shot learners.** In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on
Yuxian Gu, Xu Han, Zhiyuan Liu, and Minlie Huang. 2021. PPT: pre-trained prompt tuning for few-shot learning. CoRR, abs/2109.04332.

Nikolaus Hansen, Sibylle D. Müller, and Petros Koumoutsakos. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evol. Comput., 11(1):1–18.

Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. Evol. Comput., 9(2):159–195.

Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2021. Towards a unified view of parameter-efficient transfer learning. CoRR, abs/2110.04366.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagnet classification. In 2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015, pages 1026–1034. IEEE Computer Society.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016, pages 770–778. IEEE Computer Society.

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussiére, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for NLP. In Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, volume 97 of Proceedings of Machine Learning Research, pages 2790–2799. PMLR.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. CoRR, abs/2106.09685.

Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How can we know what language models know. Trans. Assoc. Comput. Linguistics, 8:423–438.

Kirthivasan Kandasamy, Jeff G. Schneider, and Barnabás Póczos. 2015. High dimensional bayesian optimization and bandits via additive models. In Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015, volume 37 of JMLR Workshop and Conference Proceedings, pages 295–304. JMLR.org.

Yi Mei, Mohammad Nabi Omidvar, Xiaodong Li, and Xin Yao. 2016. A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization. In Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, New York, NY, USA, June 19-21, 2016, pages 1291–1299. JMLR.org.

Benjamin Letham, Roberto Calandra, Akshara Rai, and Eytan Bakshy. 2020. Re-examining linear embeddings for high-dimensional Bayesian optimization. In Advances in Neural Information Processing Systems 33, virtual.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: denoising sequence-to-sequence pretraining for natural language generation, translation, and comprehension. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020, pages 7871–7880. Association for Computational Linguistics.
A Deviation for $\mu$ and $\sigma$ of Normal Distribution

Assume the variable $z \in \mathbb{R}^d$ is sampled from a normal distribution $\mathcal{N}(\mu_1, \sigma_1)$ that is maintained by the CMA-ES, the random projection $A \in \mathbb{R}^{D \times d}$ is generated from another normal distribution $\mathcal{N}(\mu_2, \sigma_2)$, then each entry in the projected prompts $p = Az$ is also normally distributed. Note that $p_{ij} = \sum_k A_{ik} z_{kj}$, where the probability density function (PDF) of $A_{ik} z_{kj}$ is as follows,

$$\text{PDF}(A_{ik} z_{kj}) = A \cdot \frac{1}{\sqrt{2\pi\sigma}} e^{-\frac{(x-\bar{\mu})^2}{2\bar{\sigma}^2}}, \quad (8)$$

where $\bar{\mu} = \frac{\mu_1 \sigma_2^2 + \mu_2 \sigma_1^2}{\sigma_1^2 + \sigma_2^2}, \quad (9)$
and $\bar{\sigma} = \frac{\sigma_1 \sigma_2}{\sqrt{\sigma_1^2 + \sigma_2^2}}, \quad (10)$

where $A$ is some scaling factor, $\bar{\mu}$ and $\bar{\sigma}$ are the mean and standard deviation of the new normal distribution. Assume $\{A_{ik} z_{kj}\}_{k=1}^d$ are independent variables, their sum (i.e., $p_{ij}$) is also normally distributed with

$$\hat{\mu} = d\bar{\mu} = \frac{d(\mu_1 \sigma_2^2 + \mu_2 \sigma_1^2)}{\sigma_1^2 + \sigma_2^2}, \quad (11)$$
$$\hat{\sigma} = \sqrt{d\sigma^2} = \frac{\sigma_1 \sigma_2 \sqrt{d}}{\sqrt{\sigma_1^2 + \sigma_2^2}}. \quad (12)$$

For simplicity, assume the initial normal distribution of the CMA-ES is a standard normal distribution $\mathcal{N}(0, 1)$, i.e., $\mu_1 = 0$ and $\sigma_1 = 1$, then we have

$$\hat{\mu} = \frac{d\mu_2}{1 + \sigma_2^2}, \quad (13)$$
$$\hat{\sigma} = \frac{\sigma_2 \sqrt{d}}{\sqrt{1 + \sigma_2^2}}. \quad (14)$$

Let $\hat{\mu}$ and $\hat{\sigma}$ be the observed mean and standard deviation of word embeddings (or hidden states), we can obtain $\mu_2$ and $\sigma_2$, i.e., mean and standard deviation of the desired normal distribution for generating the random projection, as follows,

$$\mu_2 = \frac{\hat{\mu}}{d - \hat{\sigma}^2}, \quad (15)$$
$$\sigma_2 = \frac{\hat{\sigma}}{\sqrt{d - \hat{\sigma}^2}}. \quad (16)$$