

# Memory-Efficient Fine-Tuning of Transformers via Token Selection

Antoine Simoulin\*, Namyong Park\*, Xiaoyi Liu, Grey Yang  
Meta AI

{antoinesimoulin,namyongp,xiaoyiliu,glyang}@meta.com

## Abstract

Fine-tuning provides an effective means to specialize pre-trained models for various downstream tasks. However, fine-tuning often incurs high memory overhead, especially for large transformer-based models, such as LLMs. While existing methods may reduce certain parts of the memory required for fine-tuning, they still require caching all intermediate activations computed in the forward pass to update weights during the backward pass. In this work, we develop **TOKENTUNE**, a method to reduce memory usage, specifically the memory to store intermediate activations, in the fine-tuning of transformer-based models. During the backward pass, **TOKENTUNE** approximates the gradient computation by backpropagating through just a subset of input tokens. Thus, with **TOKENTUNE**, only a subset of intermediate activations are cached during the forward pass. Also, **TOKENTUNE** can be easily combined with existing methods like LoRA, further reducing the memory cost. We evaluate our approach on pre-trained transformer models with up to billions of parameters, considering the performance on multiple downstream tasks such as text classification and question answering in a few-shot learning setup. Overall, **TOKENTUNE** achieves performance on par with full fine-tuning or representative memory-efficient fine-tuning methods, while greatly reducing the memory footprint, especially when combined with other methods with complementary memory reduction mechanisms. We hope that our approach will facilitate the fine-tuning of large transformers, in specializing them for specific domains or co-training them with other neural components from a larger system. Our code is available at <https://github.com/facebookresearch/tokentune>.

## 1 Introduction

Fine-tuning is an effective method for specializing large pre-trained models, either by using direct

\* Equal contribution

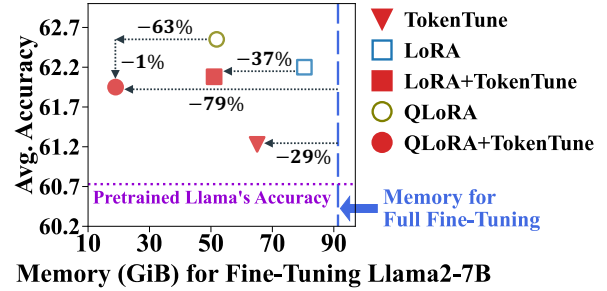


Figure 1: **TOKENTUNE** greatly reduces the GPU memory usage for fine-tuning the Llama2-7B model (e.g., using only 37% of the memory QLoRA (Dettmers et al., 2023) requires), while achieving similar accuracy to representative memory-efficient fine-tuning methods. Accuracy and memory usage numbers are listed in Table 2 and Fig. 4. See Sec. 5 for details on experiments.

supervision from the training set of a given task (Howard and Ruder, 2018; Devlin et al., 2019; Raffel et al., 2020), from curated instruction datasets (Mishra et al., 2022; Wei et al., 2022; Taori et al., 2023), or from human feedback via reinforcement learning (Ouyang et al., 2022; Bai et al., 2022; Touvron et al., 2023). However, fine-tuning is not necessarily an efficient method, especially for transformer-based large language models (LLMs), since their large number of parameters leads to large compute and memory requirements. For instance, fine-tuning GPT-3 175B (Brown et al., 2020) or LLaMA 65B (Touvron et al., 2023) typically requires 1,200 GB and 780 GB of GPU memory, as reported in Hu et al. (2022) and Dettmers et al. (2023), respectively.

GPU memory usage during fine-tuning can be broken down into three parts: storing (1) the model parameters, (2) the parameter gradients and optimizer states, and (3) the intermediate activations. Parameter-Efficient Fine-Tuning (PEFT) (Houlsby et al., 2019; Hu et al., 2022) aims at updating a small number of parameters, e.g., by optimizing a subset of the backbone model’s parameters

while freezing others, which reduces the memory requirements to store the parameters’ gradients and optimizer states. Alternatively, quantization techniques (Dettmers et al., 2022, 2023; Liu et al., 2024) use low precision data types for model parameters, which reduces the memory cost. For example, in fine-tuning the Llama2-7B model, LoRA (Hu et al., 2022) and QLoRA (Dettmers et al., 2023), which are representative PEFT and quantization-based methods, reduce the memory needed for full fine-tuning by 12% and 43%, respectively (Figure 1). However, such existing approaches still require caching all of the intermediate activations computed in the forward pass to obtain the gradients during the backward pass.

In this work, we propose a method for memory-efficient fine-tuning, named TOKENTUNE, which aims to significantly reduce the GPU memory dedicated to storing intermediate activations during the forward pass without sacrificing the model performance on various downstream tasks. To this end, TOKENTUNE selects a subset of the input tokens in the context, and fine-tunes the model with respect to those selected tokens. More specifically, during the backward pass, TOKENTUNE approximates the gradient computation by backpropagating through the selected tokens, and thus only a subset of the intermediate activations need to be cached during the forward pass, thereby reducing the memory cost.

We demonstrate the effectiveness of TOKENTUNE using both medium- and large-size language models, namely, BERT (Devlin et al., 2019) and Llama (Touvron et al., 2023), which have hundreds of millions, and billions of parameters, respectively. Overall, our results show that fine-tuning with TOKENTUNE leads to downstream task performance on par with that of full fine-tuning or representative methods for memory-efficient fine-tuning, while drastically reducing the memory footprint. Notably, TOKENTUNE can be effectively combined with existing methods, achieving a greater reduction in memory usage. For instance, by combining TOKENTUNE with QLoRA (Dettmers et al., 2023), we can fine-tune Llama2-7B using just about one third of the memory QLoRA alone requires as Figure 1 shows. To sum, our contributions are as follows.

- **Novelty.** TOKENTUNE, to the best of our knowledge, is the first method that reduces GPU memory usage for fine-tuning via token selection<sup>1</sup>.

<sup>1</sup>A preliminary version of this work was presented at a non-archival workshop (Simoulin et al., 2023).

- **Combinability.** TOKENTUNE can be combined with existing memory-efficient fine-tuning methods, leading to further memory reduction.
- **Effectiveness.** We perform extensive experiments, showing that TOKENTUNE achieves similar accuracy to representative memory-efficient methods, while greatly reducing the memory footprint during fine-tuning, e.g., using only 21% of what full fine-tuning requires (Figure 1).

## 2 Related Work

### 2.1 Parameter-Efficient Fine-Tuning (PEFT)

PEFT methods, which aim to limit the computing resources for fine-tuning LLMs, can be divided into four categories (Han et al., 2024; Xu et al., 2023).

**Selective PEFT** methods update only a subset of the backbone model parameters using weight masking strategies, such as learnable binary masking (Guo et al., 2021) and parameter importance estimation using Fisher information (Sung et al., 2021; Das et al., 2023). Other selective PEFT methods focus on updating specific modules, e.g., the cross-attention layers (Gheini et al., 2021) and the bias terms (Zaken et al., 2022; Lawton et al., 2023).

**Additive PEFT** methods add a few parameters to the frozen pre-trained model, and fine-tune only the added parameters. E.g., adapters inject small layers within the transformer block, either sequentially after its sublayers (Houlsby et al., 2019; Pfeiffer et al., 2021), or as a side network running in parallel to the sublayers (He et al., 2022a; Zhu et al., 2021). Alternatively, soft prompt-based approaches (Li and Liang, 2021; Qin and Eisner, 2021; Liu et al., 2022) prepend continuous learnable vectors to the input of a frozen model and tune them for each task.

**Reparameterized PEFT** methods perform low-rank transformation, utilizing the low intrinsic dimension of LLMs (Aghajanyan et al., 2021). LoRA (Hu et al., 2022) is the most representative approach, where an update to the model weights is captured via its low-rank decomposition. Several studies followed to improve LoRA, e.g., to support dynamic rank selection (Valipour et al., 2023; Zhang et al., 2023b), and to address overfitting (Lin et al., 2024) and overconfidence (Yang et al., 2024).

**Hybrid PEFT** methods aim to combine different PEFT approaches, e.g., adapters, prefix-tuning, and LoRA. The design space of combinations of PEFT

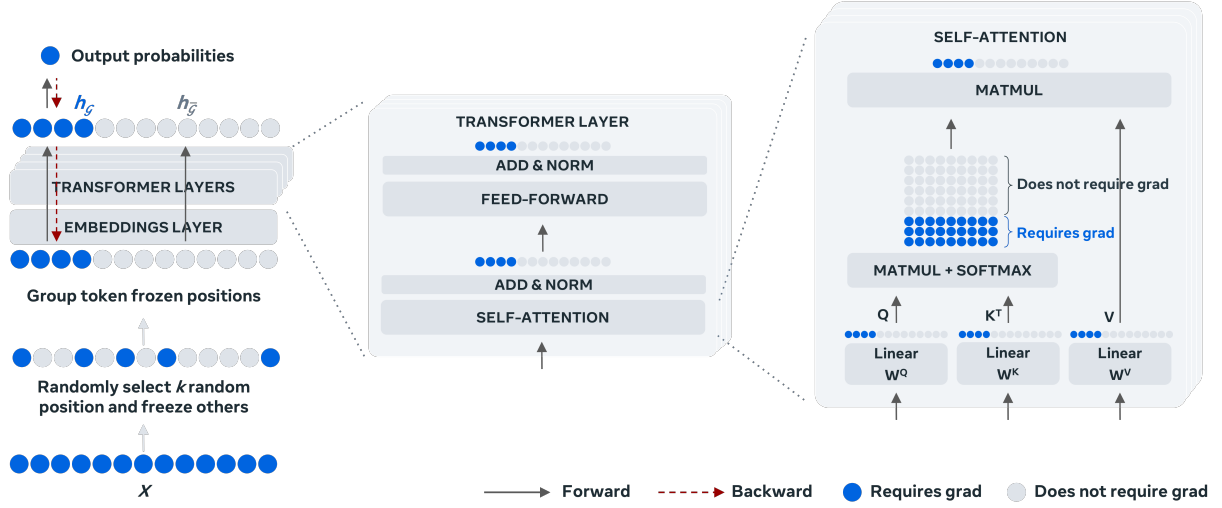


Figure 2: TOKENTUNE achieves memory-efficient fine-tuning of transformers via token selection. During the backward pass, we compute the gradient for only a subset of  $k$  input tokens, while the others are frozen (in gray in the figure). During the forward pass, all input positions are used, but only a subset of the activations is cached in memory (in blue in the figure). TOKENTUNE is applicable to various transformer-based models, as well as different language modeling tasks, as our experiments with BERT (Devlin et al., 2019) and Llama (Touvron et al., 2023) show.

methods has been explored either manually (He et al., 2022a; Mao et al., 2022), or automatically, e.g., by leveraging neural architecture search methods (Zhang et al., 2022b; Zhou et al., 2024).

While the above PEFT methods effectively improve parameter efficiency, they may still incur significant memory overhead during fine-tuning (Sung et al., 2022; Jin et al., 2023). The proposed TOKENTUNE can be combined with these PEFT methods, enabling them to achieve both parameter and memory efficiency, as Sections 4 and 5 show.

## 2.2 Memory-Efficient Fine-Tuning

There exist several techniques that can be used to improve the memory efficiency in fine-tuning LLMs, which we organize into four groups.

**Memory-Efficient PEFT.** Some PEFT methods aim to achieve memory and parameter efficiency simultaneously. Side tuning methods (Zhang et al., 2020; Sung et al., 2022) introduce small learnable side networks separated from the backbone model, and channel backpropagation only through the side networks, thereby reducing the memory requirements for gradients and intermediate activations. By utilizing the reversible model, MEFT (Liao et al., 2023) avoids the need to cache intermediate activations in the forward pass. LoRA-FA (Zhang et al., 2023a) improves LoRA by addressing its high memory usage for input activations via freez-

ing LoRA’s down-projection weights.

**Gradient Checkpointing** (Chen et al., 2016; Gruslys et al., 2016) reduces the memory requirement for model training by storing only a subset of intermediate activations in the forward pass, and recomputing the others during the backward pass.

**Quantization** is a compression technique that reduces the number of bits for storing numerical values. With quantization, parameters are represented with lower-precision data types (Dettmers et al., 2022, 2023; Liu et al., 2024), leading to memory reduction in both fine-tuning and inference.

**Approximate Gradient Methods** reduce the memory usage by avoiding the exact gradient computation involved with full fine-tuning, and instead using an approximate estimate of the gradient for weight updates. To this end, a few methods employ low-rank factorization, where they reduce memory cost by utilizing the low-rank structure of the gradients (Zhao et al., 2024) or the second-order statistics (Shazeer and Stern, 2018). Alternatively, MeZO (Malladi et al., 2023) approximates the gradient using only forward passes, building upon the zeroth-order optimization technique (Spall, 1992).

The proposed TOKENTUNE can be considered an approximate gradient method, as its token-selective fine-tuning strategy leads to an approximation of the full gradient, which is a completely new di-

rection investigated to improve memory efficiency in fine-tuning. Also, being complementary to prior methods, TOKENTUNE can be combined with them, resulting in further memory reduction.

### 3 TOKENTUNE

Previous studies analyzing the structure of the sparsity of activations and gradients (Kurtz et al., 2020; Liu et al., 2023; Dai et al., 2022) suggest that some neurons and activations could have a predominant importance, while some others may have smaller contributions to the loss and output computation. Inspired by these works, we hypothesize that for many downstream tasks, not all tokens in the sequence would need to be involved in the fine-tuning—more specifically, backpropagation—of transformer models. Instead, we conjecture that, when restricted to backpropagating through a subset of tokens, transformers could be further optimized for the downstream task by enabling the additional learning and adjustments, which need to happen during the fine-tuning for the given task, to be done in a more compact way, i.e., by incorporating the additional knowledge more succinctly with respect to the selected subset of tokens.

Figure 2 illustrates TOKENTUNE, aiming at reducing the memory needed to store the intermediate activations used for gradient computation. Given an input sequence  $X$ , a transformer associates each token from the input sequence to an embedding and computes a corresponding sequence of hidden states  $h$  through multiple layer applications. For each input sequence, we select  $k$  random positions.<sup>2</sup> We organize each layer’s input in two groups, one with the  $k$  selected input positions,  $h_{\mathcal{G}}$ , and the other with the remaining un-selected positions,  $h_{\bar{\mathcal{G}}}$ , such that  $h = [h_{\mathcal{G}}, h_{\bar{\mathcal{G}}}]$ , with  $[\ ]$  denoting the concatenation operator and  $|\mathcal{G}| = k$ . The re-ordering does not impact the computation as the position is directly encoded in the hidden states. With this token selection scheme, the classification objective  $\mathcal{L}_{\text{CLS}}$  and the language modeling objective  $\mathcal{L}_{\text{LM}}$  used by TOKENTUNE are as follows.

**Classification Task.** The goal is to assign the right class or label  $y$  for the given sequence. Given the hidden states from the transformer layers, we use the average of the hidden states from the  $k$  selected positions of the last layer as input for an

MLP, which outputs a probability distribution over the classes of the task, as given by Eq. 1. During the evaluation, we use the average from all hidden states of the last layer as input for the MLP.

$$\begin{aligned}\pi &= \text{MLP} \left( \frac{1}{k} \sum_{i \in \mathcal{G}} h_i \right) \\ p(y|X) &= \text{softmax}(\pi) \\ \mathcal{L}_{\text{CLS}} &= -\log p(y|X)\end{aligned}\tag{1}$$

**Language Modeling Task.** The goal is to learn the probability distribution of a token, given all preceding tokens. We train the language model by applying the traditional cross-entropy loss to the set of  $k$  randomly selected positions as given by Eq. 2 below, with  $W_{\text{lm}}$  denoting the head projecting the hidden state back into the vocabulary dimension.

$$\begin{aligned}p(x_i|x_{<i}) &= \text{softmax}(h_i W_{\text{lm}}) \\ \mathcal{L}_{\text{LM}} &= -\sum_{i \in \mathcal{G}} \log P(x_i|x_{<i})\end{aligned}\tag{2}$$

The key element of our method is that we disable the gradient computation for the un-selected tokens in  $\bar{\mathcal{G}}$ . Thus, only the  $k$  selected tokens in  $\mathcal{G}$  contribute to the gradient computation during the backward pass. We detail the method in the case of dense layers and attention mechanism in Section 3.1 and Section 3.2, respectively.

#### 3.1 TOKENTUNE for Dense and Normalization Layers

We consider a dense layer  $a = \sigma(z) = \sigma(hW + b)$  with weight  $W$ , bias  $b$ , nonlinear function  $\sigma$ , input  $h$ , pre-activation  $z$ , and output  $a$ . Eq. 3 computes the gradient with respect to  $W$  and  $b$  when backpropagating a loss  $\mathcal{L}$  through the layer:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial W} &= \frac{\partial \mathcal{L}}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial W} = \frac{\partial \mathcal{L}}{\partial a} \sigma' h \\ \frac{\partial \mathcal{L}}{\partial b} &= \frac{\partial \mathcal{L}}{\partial a} \frac{\partial a}{\partial z} \frac{\partial z}{\partial b} = \frac{\partial \mathcal{L}}{\partial a} \sigma'\end{aligned}\tag{3}$$

If we backpropagate the error only through the selected tokens in  $\mathcal{G}$ , and disable the gradient computation for the unselected positions in  $\bar{\mathcal{G}}$ , we have:

$$\frac{\partial \mathcal{L}}{\partial a} = \left[ \frac{\partial \mathcal{L}}{\partial a_{\mathcal{G}}}, \frac{\partial \mathcal{L}}{\partial a_{\bar{\mathcal{G}}}} \right] = \left[ \frac{\partial \mathcal{L}}{\partial a_{\mathcal{G}}}, 0 \right]\tag{4}$$

Plugging that into Eq. 3, we have:

$$\frac{\partial \mathcal{L}}{\partial W} = \left[ \frac{\partial \mathcal{L}}{\partial a_{\mathcal{G}}} \sigma' h_{\mathcal{G}}, 0 \right]; \quad \frac{\partial \mathcal{L}}{\partial b} = \left[ \frac{\partial \mathcal{L}}{\partial a_{\mathcal{G}}} \sigma', 0 \right]\tag{5}$$

<sup>2</sup>We select the positions using a uniform distribution. However, we always include the [CLS] token—a special symbol prepended as the beginning of every input sentence.



Given Eq. 5, we only need to cache  $h_G$  for applying the chain rule, instead of the full activation  $h$ .

Regarding implementation, we use Algorithm 1 which explicitly splits the hidden states into two groups where  $h_G$  corresponds to the tokens selected to be fine-tuned and  $h_{\bar{G}}$  corresponds to the unselected tokens. As shown in Eq. 6 and Eq. 7, the forward pass is identical to standard fine-tuning except that we disable the gradient computation for the positions for  $h_{\bar{G}}$  in Eq. 7 with the context "torch.no\_grad()" in PyTorch.

$$h_G = h_G W + b \quad (6)$$

$$h_{\bar{G}} = h_{\bar{G}} W + b \quad (7)$$

where  $W$  denotes the weights  $W_1$  and  $W_2$  for the feed-forward layers. We apply the same methodology for normalization layers.

### 3.2 TOKENTUNE for Attention Layers

For attention layers, we compute the attention as:

$$[Q_G, K_G, V_G] = h_G W_{[Q,K,V]} + b_{[Q,K,V]} \quad (8)$$

$$[Q_{\bar{G}}, K_{\bar{G}}, V_{\bar{G}}] = h_{\bar{G}} W_{[Q,K,V]} + b_{[Q,K,V]} \quad (9)$$

$$h_G = \text{softmax} \left( \frac{Q_G [K_{\bar{G}}, K_G]^T}{\sqrt{d}} \right) [V_{\bar{G}}, V_G] \quad (10)$$

$$h_{\bar{G}} = \text{softmax} \left( \frac{Q_{\bar{G}} [K_{\bar{G}}, K_G]^T}{\sqrt{d}} \right) [V_{\bar{G}}, V_G] \quad (11)$$

where  $W_{[Q,K,V]} \in \mathbb{R}^{d \times 3d}$  denotes the concatenated weights for the queries, keys, and values. For the computation of un-selected positions in Eq. 9 and Eq. 11, we again disable the gradient computation in PyTorch. Algorithm 1 illustrates the steps for the forward pass of a transformer model with the proposed TOKENTUNE algorithm described in Sections 3.1 and 3.2.

## 4 Application to Medium-Size Encoders

Alternative methods such as zero-shot learning or prompting usually underperform fine-tuning (Brown et al., 2020). Thus, in many cases, fine-tuning medium size language models may offer a better balance in terms of cost and performance, compared with fine-tuning large language models (LLMs) or conditioning their outputs with prompt approaches (Li et al., 2022; Schick and Schütze, 2021). Medium-size models may also be used as individual components, co-trained to encode information for a larger system (Pfeiffer et al., 2023). Finally, as detailed in Appendix E, the distribution of the GPU memory usage may be very different given the order of magnitude of the fine-tuned

---

**Algorithm 1:** TOKENTUNE (We omit layer normalization, skip connections, non-linear functions, and multi-head attention for simplicity)

---

**Input:** input sequence  $X$

**Output:**  $h_G, h_{\bar{G}}$

---

```

1 Compute input token embeddings  $h$ 
2 Re-organize input tokens into two groups ( $h_G$  and  $h_{\bar{G}}$ )
3 for layer in transformers' layers do
    // Compute the attention layer
4    $[Q_G, K_G, V_G] = h_G W_{[Q,K,V]} + b_{[Q,K,V]}$ 
5    $h_G = \text{softmax} \left( \frac{Q_G [K_{\bar{G}}, K_G]^T}{\sqrt{d}} \right) [V_{\bar{G}}, V_G]$ 
6   with torch.no_grad():
7      $[Q_{\bar{G}}, K_{\bar{G}}, V_{\bar{G}}] = h_{\bar{G}} W_{[Q,K,V]} + b_{[Q,K,V]}$ 
8      $h_{\bar{G}} = \text{softmax} \left( \frac{Q_{\bar{G}} [K_{\bar{G}}, K_G]^T}{\sqrt{d}} \right) [V_{\bar{G}}, V_G]$ 
    // Compute the feed-forward layer
9    $h_G = h_G W_1 + b_1$ 
10   $h_G = h_G W_2 + b_2$ 
11  with torch.no_grad():
12     $h_{\bar{G}} = h_{\bar{G}} W_1 + b_1$ 
13     $h_{\bar{G}} = h_{\bar{G}} W_2 + b_2$ 
14 Re-organize input tokens into the original order
```

---

model's number of parameters. For large-size models, the majority of the memory is often dedicated to storing parameters and optimizer states, thus maximizing the relevance of PEFT approaches. For medium-size language models, fine-tuned with large batch sizes, the majority of the memory may be dedicated to storing the intermediate activation, thus maximizing the impact of TOKENTUNE.

### 4.1 Downstream Task Performance

We first validate the relevance of our method on the GLUE benchmark (Wang et al., 2018). We use a similar hyper-parameter search space as in (Zaken et al., 2022), by performing a cross validation on the dev set using a learning rate in  $[5e^{-5}, 3e^{-5}, 2e^{-5}, 1e^{-5}]$ . We set the batch size to 16 and perform 3 epochs on large datasets and 20 epochs on small ones (MRPC, STS-B, CoLA). We use BERT-large (Devlin et al., 2019) and either fine-tune the model fully, or use TOKENTUNE and propagate the gradient through 16 input positions. We then evaluate our model on the test set and report the results in Table 1.

As shown in the second part of Table 1, the average GLUE score of TOKENTUNE is comparable to that of full fine-tuning, thus empirically validat-

Table 1: Results from BERT-large (Devlin et al., 2019) on GLUE test tasks scored using the benchmark server. We report the Matthew’s Correlation for CoLA, the Spearman correlation for STS-B, F1 score for MRPC and QQP. We report the accuracy on the MNLI matched test split and the accuracy for every other tasks. The “Param.” column indicates the ratio of the number of updated parameters for each task by the number of parameters in the backbone model. We indicate in **bold** the best result for each task. <sup>†</sup> indicates models we trained. We report adapter results from (Houlsby et al., 2019), BitFit from (Zaken et al., 2022) and Diff Pruning from (Guo et al., 2021). For LoRA (Hu et al., 2022) and Ladder Side Tuning (LST) (Sung et al., 2022), we select the best learning rate in the dev set between the values proposed in the original papers,  $[5e^{-4}, 4e^{-4}, 3e^{-4}, 2e^{-4}]$  and  $[3e^{-4}, 1e^{-3}, 3e^{-3}]$ , respectively. We do not use the initialization setup proposed in LoRA or LST nor do we drop any layers for the LST method.

Method	Param. (%)	CoLA	SST-2	MRPC	QQP	QNLI	MNLI	STS-B	Avg. <sup>†</sup>
Avg. # Tokens	—	11.3	13.3	53.2	30.6	49.4	39.8	27.8	32.2
Full Fine-Tuning <sup>†</sup>	100.0	60.7	<b>94.6</b>	88.3	<b>72.0</b>	92.4	85.8	85.8	82.8
Adapters	3.6	59.5	94.0	89.5	71.8	90.7	84.9	<b>86.9</b>	82.5
BitFit	0.1	59.7	94.2	88.9	70.5	92.0	84.5	85.0	82.1
Diff Pruning	0.5	<b>61.1</b>	94.1	<b>89.7</b>	71.1	<b>93.3</b>	<b>86.4</b>	86.0	<b>83.1</b>
Ladder Side Tuning <sup>†</sup>	2.4	56.4	93.4	88.0	66.9	89.1	82.9	86.6	80.5
LoRA <sup>†</sup>	0.3	58.5	94.0	89.2	71.1	91.1	84.7	84.6	81.9
TOKENTUNE <sup>†</sup>	100.0	59.6	93.9	88.0	70.8	91.0	85.4	86.0	82.1

ing the effectiveness of our approach. Table 1 also shows that TOKENTUNE either outperforms or performs similarly to existing SOTA approaches. Precisely speaking, the performance of these memory-efficient fine-tuning methods, including TOKENTUNE, is often slightly worse than that of full fine-tuning. In comparison to full fine-tuning, some amount of performance loss with these methods is expected as they approximate or simplify the optimization process of full fine-tuning to reduce memory footprint. We hypothesize that some tasks, such as QQP and QNLI, are more difficult, or sensitive to overfitting than others, given that updating a small proportion of model parameters or using only a subset of input tokens for gradient computation achieves suboptimal performances on those tasks in most cases. The former case would require the development of sophisticated techniques to more effectively select a subset of parameters or input tokens to optimize, while the latter case may benefit from the use of regularization techniques for neural networks, including Gouk et al. (2021); Foret et al. (2021); Li and Zhang (2021), the investigation of which we leave for future studies.

## 4.2 Ratio of Tuned Input Positions

Given our token-selective fine-tuning approach, we then evaluate the impact of the number of frozen input positions on the performance. We use our selective procedure to fine-tune BERT-base on two tasks from the GLUE benchmark: MRPC and STS-

B. We set the hyper-parameters as follows:  $5e^{-5}$  for the learning rate, 32 for the batch size and 4 epochs. We use different values for  $k$  (i.e., the number of trained input positions), ranging between 4 and 64. We report in Figure 3 (right), the average performance on the dev set of the tasks.<sup>3</sup>

As seen in Figure 3, the performance increases from 84.8 to 88.8 as the number of trained positions increases from 4 to 64. However, by only tuning 32 positions, we already reach an average performance of 88.4, close to the 88.8 obtained by training 64 input positions. Our method surpasses the performance of freezing some bottom layers, as shown in (Lee et al., 2019), where only tuning the four bottom layers resulted in a 10% decrease in performance on the GLUE benchmark.

## 4.3 GPU Memory Impact

Finally, we analyze the GPU memory required to fine-tune models using various approaches. We train our BERT-base model for 100 steps on the CoLA task using various batch sizes and report the peak GPU memory used. We compare with two other PEFT fine-tuning approaches close to ours: Ladder Side Tuning (Sung et al., 2022) and LoRA (Hu et al., 2022). LoRA freezes most of the model

<sup>3</sup>We provide some descriptive statistics in Appendix F to better understand how the absolute number of frozen input positions relates with the relative number of frozen input positions. The statistics include distribution of the sentence length for the two subtasks (MRPC and STS-B) used to produce Figure 3 (right).

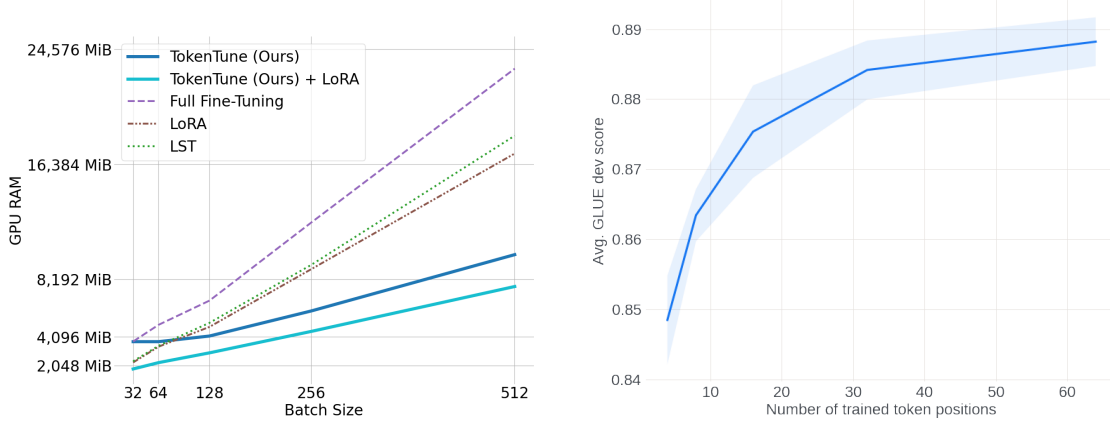


Figure 3: (left) We plot the GPU memory required to train BERT-base on the CoLA task given varying batch sizes. We compare our approach with two PEFT approaches: Ladder Side Tuning (LST) and LoRA. (right) We plot the mean and standard deviation performance on the dev set of five runs when training BERT-base on two tasks from the GLUE benchmark: MRPC and STS-B. We use our memory efficient fine-tuning approach with a different number of selected input tokens for the gradient computation.

parameters, while only training additional low-rank matrices, whose weights are added to the backbone network. Ladder Side Tuning (LST) freezes the model parameters but trains a side-network with smaller dimensions, taking as input intermediate activations from the backbone model.

Figure 3 shows the evolution of the required GPU memory with respect to the batch size. GPU memory increases with the batch size for every approach. TOKENTUNE is more memory efficient by a large margin. When using a batch size of 512, it requires two times less memory than full fine-tuning: 23,196 MiB needed for full fine-tuning is reduced to 9,952 MiB with our method.

All methods minimize GPU memory usage. LoRA and LST reduce the memory required to store optimizer states and parameter gradients, while our method reduces the memory for storing intermediate activations. Interestingly enough, it is possible to use these approaches in conjunction to reduce the memory for all three contributions. Fig. 3 shows that we can further reduce the memory by combining TOKENTUNE with LoRA, thus requiring only 7,682 MiB with a batch size of 512, a third of the memory used for full fine-tuning.

## 5 Application to Large-Size Decoders

We also seek to evaluate our method on larger size pre-trained language models (LLMs).

### 5.1 Instruction Tuning and Few-Shot Evaluation

LLMs are typically further fine-tuned on curated datasets to tailor them to specific domains and enhance their capacity to follow instructions (Wang et al., 2023; Taori et al., 2023; Mukherjee et al., 2023). In this section, we employ instruction tuning on these datasets to fine-tune the LLMs and then assess the performance of the resulting models using few-shot benchmarks.

**Instruction Tuning.** We fine-tune the Llama2-7B model (Touvron et al., 2023) via instruction tuning with the Open-Platypus<sup>4</sup> (Lee et al., 2023) dataset. Note that, while Open-Platypus consists of 11 open-source datasets, we exclude two of them<sup>5</sup> that include outputs from GPT (OpenAI, 2023), and instead use the other nine datasets for fine-tuning.

**Hyper-Parameter Settings.** We conduct all experiments in this section on Nvidia H100 GPU. Following Lee et al. (2023), we fine-tune the model for one epoch, and use a learning rate of  $4e^{-4}$  for LoRA (Hu et al., 2022) and QLoRA (Dettrmers et al., 2023), and  $4e^{-5}$  otherwise. We use a batch size of 1 with 32 gradient accumulation steps. We apply the adapters on the feed-forward modules from each layer, following the method described in He et al. (2022b). We prompt the model without

<sup>4</sup><https://huggingface.co/datasets/garage-bAInd/Open-Platypus>

<sup>5</sup>leetcodesolutions-python-testgen-gpt4 and airoboros-gpt4-1.4.1

Table 2: Few-shot evaluation on question-answering benchmarks including: AI2 Reasoning Challenge (25-shot) (Clark et al., 2018), MMLU (5-shot) (Hendrycks et al., 2021), HellaSwag (10-shot) (Zellers et al., 2019), TruthfulQA (0-shot) (Lin et al., 2022), and WinoGrande (0-shot) (Sakaguchi et al., 2020). We use the evaluation scripts and prompt formatting from the "Language Model Evaluation Harness" (Gao et al., 2021). We report the average accuracy on five MMLU ethics tasks and WinoGrande, the normed accuracy on ARC and HellaSwag, and the MC2 score on TruthfulQA. We indicate in **bold** the best result for each task. We report the results with the raw Llama2-7B model (Touvron et al., 2023) and the Llama2-7B fine-tuned on the Platypus curated instruction dataset (Lee et al., 2023) using LoRA (Hu et al., 2022), QLoRA (Dettmers et al., 2023) and the proposed TOKENTUNE. When fine-tuning with TOKENTUNE, we select 30% of the tokens for the gradient computation.

Method	MMLU	ARC	Hella Swag	Truthful QA	Wino Grande	Avg. $\uparrow$
Llama 7B	64.44	52.39	<b>78.97</b>	38.97	68.90	60.73
Llama 7B w/ LoRA	<b>65.89</b>	55.38	78.76	42.64	68.35	62.20
Llama 7B w/ LoRA+TOKENTUNE (Ours)	65.42	54.01	78.82	<b>43.78</b>	68.35	62.08
Llama 7B w/ QLoRA	65.08	<b>56.06</b>	78.60	43.64	69.38	<b>62.55</b>
Llama 7B w/ QLoRA+TOKENTUNE (Ours)	65.78	53.92	78.74	41.91	69.38	61.95
Llama 7B w/ TOKENTUNE (Ours)	63.06	53.07	77.90	42.18	<b>69.93</b>	61.23

step-wise reasoning using the Alpaca (Taori et al., 2023) prompt template detailed in Appendix A.

**Few-Shot Evaluation.** Then, we evaluate our method against other memory-efficient fine-tuning approaches by assessing its performance on several few-shot benchmarks, such as MMLU (Hendrycks et al., 2021), ARC easy and challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), TruthfulQA (Lin et al., 2022), and WinoGrande (Sakaguchi et al., 2020). We utilize the evaluation scripts provided by the "Language Model Evaluation Harness" (Gao et al., 2021). During the evaluation process, the model outputs the probability associated with closed-form problems defined by the context, question, and multiple potential answers. We select the answer choice with the text associated with the highest probability.

Table 2 reports the accuracy of the model output against the ground truth answer. Our method achieves competitive performance gains that are comparable to the performance improvements obtained by other memory efficient fine-tuning approaches. We are able to improve the evaluation accuracy upon the base LLama2-7B model, increasing the average accuracy from 60.7 to 61.2. We observe the most significant improvements for TruthfulQA (+3.2) and WinoGrande (+1.0) tasks. We also combine TOKENTUNE with LoRA and QLoRA, further improving the evaluation accuracy compared to the use of TOKENTUNE alone.

## 5.2 Ratio of Tuned Input Positions

As done for medium-size encoders in Section 4.2, we then evaluate the impact of the ratio of tuned input positions on the few-shot accuracy. We measure the few-shot accuracy of Llama2-7B models fine-tuned using TOKENTUNE with varying ratio of tuned input positions. Table 3 shows few-shot evaluation accuracy of Llama2-7B when the ratio of fine-tuned positions ranges from 10% to 50%.

Contrary to what we observed in Section 4.2, we do not necessarily observe a strong correlation between the few-shot accuracy and the ratio of tuned positions. In fact, we obtain the best performances most often when 20%–30% of input positions are fine-tuned. It is important to observe that the average sequence length in these experiments far exceeds the one from the experiments on the GLUE benchmark. This suggests that tuning a relatively small number of positions may be sufficient to successfully fine-tune the model on specific datasets.

## 5.3 GPU Memory Impact

As in Section 4.3, we analyze the impact of our method on the GPU memory required to fine-tune large language models. Figure 4 and Table 3 report the GPU memory usage for fine-tuning Llama2-7B as the number of trained input tokens changes. Given an input sequence of length 2,048, Figure 4 shows that our model reduces the memory usage by up to 28%, from 89 GiB to 64 GiB when reducing the number of trained positions from 2,046 to 256.



Table 3: Few-shot evaluation results and peak memory usage (GiB) as Llama2-7B is fine-tuned on instruction datasets with (a) TOKENTUNE, (b) TOKENTUNE + LoRA and (c) TOKENTUNE + QLoRA, varying the selection ratio of input tokens. Best results in **bold**.

(a) TOKENTUNE							
Selection Ratio	Peak Mem.	MMLU	ARC	Hella Swag	Truthful QA	Wino Grande	Avg. Perf.
10%	<b>64.40</b>	61.56	51.71	78.35	41.88	70.01	60.70
20%	65.08	<b>65.01</b>	52.65	<b>78.37</b>	42.02	69.46	<b>61.50</b>
30%	65.94	63.06	<b>53.07</b>	77.90	<b>42.18</b>	69.93	61.23
40%	68.42	63.78	52.90	77.90	41.45	<b>70.32</b>	61.27
50%	74.32	62.98	52.73	78.32	42.11	69.38	61.10

(b) TOKENTUNE + LoRA							
Selection Ratio	Peak Mem.	MMLU	ARC	Hella Swag	Truthful QA	Wino Grande	Avg. Perf.
10%	<b>45.47</b>	64.17	<b>54.44</b>	78.68	38.77	<b>69.61</b>	61.13
20%	48.21	65.41	54.35	<b>79.01</b>	42.21	69.38	62.07
30%	52.77	65.42	54.01	78.82	<b>43.78</b>	68.35	62.08
40%	56.31	64.35	52.65	78.69	41.05	68.90	61.13
50%	64.34	<b>65.87</b>	54.01	78.68	42.46	69.38	<b>62.08</b>

(c) TOKENTUNE + QLoRA							
Selection Ratio	Peak Mem.	MMLU	ARC	Hella Swag	Truthful QA	Wino Grande	Avg. Perf.
10%	<b>11.47</b>	63.54	54.18	78.58	39.79	68.98	61.02
20%	15.68	64.05	53.92	<b>78.81</b>	40.33	<b>69.85</b>	61.39
30%	19.71	<b>65.78</b>	53.92	78.74	41.91	69.38	<b>61.95</b>
40%	24.11	64.85	<b>54.35</b>	78.70	<b>41.98</b>	69.14	61.80
50%	31.06	65.29	53.75	78.70	40.63	69.06	61.49

The advantage of the proposed method is that it can be combined with other memory saving methods. We measure the peak memory required to fine-tune Llama2-7B when combining TOKENTUNE with LoRA or QLoRA. Since these approaches target different parts of the memory footprint, we observe cumulative savings when they are used together. When combining LoRA with TOKENTUNE, the peak memory ranges between 78 GiB to 45 GiB depending on the number of tuned positions. Similarly, when combining QLoRA with TOKENTUNE, the peak memory decreases from 49 GiB to 12 GiB as a smaller selection ratio is used.

Overall, Figure 4 and Table 3 show that the performance of TokenTune is not very sensitive to the choice of token selection ratio, while the memory cost is significantly reduced with a smaller token selection ratio. Based on these results, our recommendation is to use 20%–30% as the default token selection ratio, and test if further improvements in performance and memory usage can be obtained for the given task, with a smaller selection ratio.

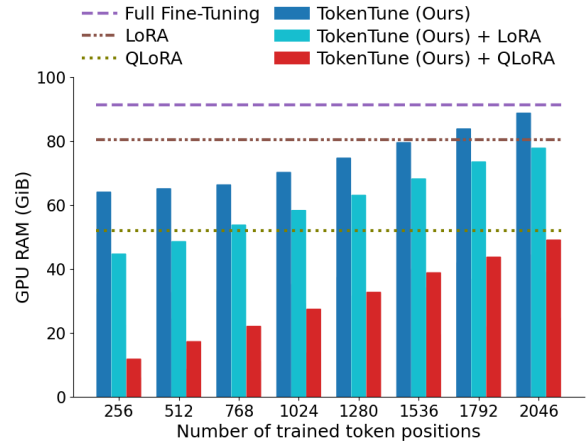


Figure 4: GPU memory required to fine-tune Llama2-7B (Touvron et al., 2023). We measure the memory by fine-tuning the model on artificially generated data with a given sequence length and batch size. We set the batch size to 1 and the sequence length to 2,048. We show the memory usage when combining TOKENTUNE with LoRA and QLoRA and plot the evolution of the memory required to fine-tune the model on a H100 GPU with a number of trained positions ranging between 256 and 2,046 (we leave at least 2 positions not tuned). Since we could not perform full fine-tuning on our hardware, we estimate the full fine-tuning memory based on the memory reported for TOKENTUNE and LoRA. Specific memory usage values can be found in Table 4.

## 6 Conclusion

In this paper, we propose TOKENTUNE, a method for reducing the GPU memory required to fine-tune transformer-based models, such as large language models. Our contributions are as follows.

- **Novelty.** TOKENTUNE is the first approach that reduces the GPU memory footprint for fine-tuning via token selection, which selects a subset of the input positions through which the gradient is propagated, while keeping the others frozen.
- **Combinability.** The proposed token selection strategy can be combined with other memory- and parameter-efficient fine-tuning approaches, achieving a greater memory reduction together.
- **Effectiveness.** We empirically benchmark TOKENTUNE using large language models with up to billions of parameters. As Figure 1 and Table 1 show, TOKENTUNE achieves similar prediction accuracy to representative memory- and parameter-efficient methods, such as LoRA and QLoRA, while significantly reducing the memory usage for fine-tuning (e.g., a joint application of TOKENTUNE and QLoRA uses 79% less memory than full fine-tuning).

## 7 Limitations

While TOKENTUNE effectively reduces the memory required for storing intermediate activations, it does not affect the other parts of GPU memory usage, such as the one for parameter gradients. However, as we showed in experiments, TOKENTUNE can be combined with memory-efficient methods that reduce those other parts of memory footprint. Also, the evaluation of TOKENTUNE in this work focused on one domain, namely, language models. Given the applicability of TOKENTUNE to other domains, such as vision (Dosovitskiy et al., 2021), we hope to investigate its effectiveness in broader settings in the future.

**Potential Risks.** Since this paper presents a method for memory-efficient fine-tuning of transformer-based models, such as LLMs, and is not tied to particular applications, we do not see potential risks of the proposed method.

## 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.
- Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom B. Brown, Jack Clark, Sam McCandlish, Chris Olah, Benjamin Mann, and Jared Kaplan. 2022. [Training a helpful and harmless assistant with reinforcement learning from human feedback](#). *CoRR*, abs/2204.05862.
- Tom B. 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*.
- Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. 2016. [Training deep nets with sublinear memory cost](#). *CoRR*, abs/1604.06174.
- Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. [Think you have solved question answering? try arc, the AI2 reasoning challenge](#). *CoRR*, abs/1803.05457.
- Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. 2022. [Knowledge neurons in pretrained transformers](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 8493–8502. Association for Computational Linguistics.
- Sarkar Snigdha Sarathi Das, Haoran Zhang, Peng Shi, Wenpeng Yin, and Rui Zhang. 2023. [Unified low-resource sequence labeling by sample-aware dynamic sparse finetuning](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 6998–7010. Association for Computational Linguistics.
- Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. 2022. [GPT3.int8\(\): 8-bit matrix multiplication for transformers at scale](#). In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.
- Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. [QLoRA: Efficient fine-tuning of quantized llms](#). In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.
- 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.
- Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. [An image is worth 16x16 words: Transformers for image recognition at scale](#). In *9th International Conference*

- on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net.
- Pierre Foret, Ariel Kleiner, Hossein Mobahi, and Behnam Neyshabur. 2021. [Sharpness-aware minimization for efficiently improving generalization](#). In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.
- Leo Gao, Jonathan Tow, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Kyle McDonell, Niklas Muennighoff, Jason Phang, Laria Reynolds, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2021. [A framework for few-shot language model evaluation](#).
- Mozhdeh Gheini, Xiang Ren, and Jonathan May. 2021. [Cross-attention is all you need: Adapting pretrained transformers for machine translation](#). In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 1754–1765. Association for Computational Linguistics.
- Henry Gouk, Timothy M. Hospedales, and Massimiliano Pontil. 2021. [Distance-based regularisation of deep networks for fine-tuning](#). In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.
- Audrunas Gruslys, Rémi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. 2016. [Memory-efficient backpropagation through time](#). In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4125–4133.
- Demi Guo, Alexander M. Rush, and Yoon Kim. 2021. [Parameter-efficient transfer learning with diff pruning](#). 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 4884–4896. Association for Computational Linguistics.
- Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. 2024. [Parameter-efficient fine-tuning for large models: A comprehensive survey](#). *CoRR*, abs/2403.14608.
- Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022a. [Towards a unified view of parameter-efficient transfer learning](#). In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.
- Junxian He, Chunting Zhou, Xuezhe Ma, Taylor Berg-Kirkpatrick, and Graham Neubig. 2022b. [Towards a unified view of parameter-efficient transfer learning](#). In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.
- Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. [Measuring massive multitask language understanding](#). In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.
- Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin de Laroussilhe, 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.
- Jeremy Howard and Sebastian Ruder. 2018. [Universal language model fine-tuning for text classification](#). In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, pages 328–339. Association for Computational Linguistics.
- Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. [LoRA: Low-rank adaptation of large language models](#). In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.
- Feihu Jin, Jiajun Zhang, and Chengqing Zong. 2023. [Parameter-efficient tuning for large language model without calculating its gradients](#). In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 321–330. Association for Computational Linguistics.
- Mark Kurtz, Justin Kopinsky, Rati Gelashvili, Alexander Matveev, John Carr, Michael Goin, William M. Leiserson, Sage Moore, Nir Shavit, and Dan Alistarh. 2020. [Inducing and exploiting activation sparsity for fast inference on deep neural networks](#). In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 5533–5543. PMLR.
- Neal Lawton, Anoop Kumar, Govind Thattai, Aram Galstyan, and Greg Ver Steeg. 2023. [Neural architecture search for parameter-efficient fine-tuning of large pre-trained language models](#). In *Findings of the Association for Computational Linguistics: ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 8506–8515. Association for Computational Linguistics.
- Ariel N. Lee, Cole J. Hunter, and Nataniel Ruiz. 2023. [Platypus: Quick, cheap, and powerful refinement of llms](#). In *NeurIPS 2023 Workshop on Instruction Tuning and Instruction Following*.



- Jaeeun Lee, Raphael Tang, and Jimmy Lin. 2019. [What would elsa do? freezing layers during transformer fine-tuning](#). *CoRR*, abs/1911.03090.
- Dongyue Li and Hongyang R. Zhang. 2021. [Improved regularization and robustness for fine-tuning in neural networks](#). In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 27249–27262.
- Xiang Lisa Li and Percy Liang. 2021. [Prefix-tuning: Optimizing continuous prompts for generation](#). 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 4582–4597. Association for Computational Linguistics.
- Xuechen Li, Florian Tramèr, Percy Liang, and Tatsunori Hashimoto. 2022. [Large language models can be strong differentially private learners](#). In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.
- Baohao Liao, Shaomu Tan, and Christof Monz. 2023. [Make pre-trained model reversible: From parameter to memory efficient fine-tuning](#). In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.
- Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. [Truthfulqa: Measuring how models mimic human falsehoods](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 3214–3252. Association for Computational Linguistics.
- Yang Lin, Xinyu Ma, Xu Chu, Yujie Jin, Zhibang Yang, Yasha Wang, and Hong Mei. 2024. [LoRA dropout as a sparsity regularizer for overfitting control](#). *CoRR*, abs/2404.09610.
- Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. [P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 61–68, Dublin, Ireland. Association for Computational Linguistics.
- Zechun Liu, Barlas Oguz, Changsheng Zhao, Ernie Chang, Pierre Stock, Yashar Mehdad, Yangyang Shi, Raghuraman Krishnamoorthi, and Vikas Chandra. 2024. [LLM-QAT: data-free quantization aware training for large language models](#). In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 467–484. Association for Computational Linguistics.
- Zichang Liu, Jue Wang, Tri Dao, Tianyi Zhou, Binhang Yuan, Zhao Song, Anshumali Shrivastava, Ce Zhang, Yuandong Tian, Christopher Ré, and Beidi Chen. 2023. [Deja vu: Contextual sparsity for efficient llms at inference time](#). In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 22137–22176. PMLR.
- Sadhika Malladi, Tianyu Gao, Eshaan Nichani, Alex Damian, Jason D. Lee, Danqi Chen, and Sanjeev Arora. 2023. [Fine-tuning language models with just forward passes](#). In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.
- Yuning Mao, Lambert Mathias, Rui Hou, Amjad Almahairi, Hao Ma, Jiawei Han, Scott Yih, and Madian Khabsa. 2022. [UniPELT: A unified framework for parameter-efficient language model tuning](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 6253–6264. Association for Computational Linguistics.
- Swaroop Mishra, Daniel Khoshnab, Chitta Baral, and Hannaneh Hajishirzi. 2022. [Cross-task generalization via natural language crowdsourcing instructions](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 3470–3487. Association for Computational Linguistics.
- Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Hassan Awadallah. 2023. [Orca: Progressive learning from complex explanation traces of GPT-4](#). *CoRR*, abs/2306.02707.
- OpenAI. 2023. [GPT-4 technical report](#). *CoRR*, abs/2303.08774.
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. [Training language models to follow instructions with human feedback](#). In *NeurIPS*.
- Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. [Adapterfusion: Non-destructive task composition for transfer learning](#). In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, pages 487–503. Association for Computational Linguistics.



- Jonas Pfeiffer, Sebastian Ruder, Ivan Vulic, and Edoardo M. Ponti. 2023. [Modular deep learning](#). *Trans. Mach. Learn. Res.*, 2023.
- Guanghui Qin and Jason Eisner. 2021. [Learning how to ask: Querying lms with mixtures of soft prompts](#). In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 5203–5212. Association for Computational Linguistics.
- Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. [Exploring the limits of transfer learning with a unified text-to-text transformer](#). *J. Mach. Learn. Res.*, 21:140:1–140:67.
- Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2020. [Winogrande: An adversarial winograd schema challenge at scale](#). In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 8732–8740. AAAI Press.
- Timo Schick and Hinrich Schütze. 2021. [It’s not just size that matters: Small language models are also few-shot learners](#). In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, pages 2339–2352. Association for Computational Linguistics.
- Noam Shazeer and Mitchell Stern. 2018. [Adafactor: Adaptive learning rates with sublinear memory cost](#). In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4603–4611. PMLR.
- Antoine Simoulin, Namyoung Park, Xiaoyi Liu, and Grey Yang. 2023. [Memory-efficient selective fine-tuning](#). In *Workshop on Efficient Systems for Foundation Models @ ICML2023*.
- James C. Spall. 1992. [Multivariate stochastic approximation using a simultaneous perturbation gradient approximation](#). *IEEE Transactions on Automatic Control*, 37:332–341.
- Yi-Lin Sung, Jaemin Cho, and Mohit Bansal. 2022. [LST: Ladder side-tuning for parameter and memory efficient transfer learning](#). In *NeurIPS*.
- Yi-Lin Sung, Varun Nair, and Colin Raffel. 2021. [Training neural networks with fixed sparse masks](#). In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 24193–24205.
- Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. [https://github.com/tatsu-lab/stanford\\_alpaca](https://github.com/tatsu-lab/stanford_alpaca).
- Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton-Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. [Llama 2: Open foundation and fine-tuned chat models](#). *CoRR*, abs/2307.09288.
- Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. 2023. [DyLoRA: Parameter-efficient tuning of pre-trained models using dynamic search-free low-rank adaptation](#). In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2023, Dubrovnik, Croatia, May 2-6, 2023*, pages 3266–3279. Association for Computational Linguistics.
- Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. [GLUE: A multi-task benchmark and analysis platform for natural language understanding](#). In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium. Association for Computational Linguistics.
- Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. [Self-instruct: Aligning language models with self-generated instructions](#). In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13484–13508. Association for Computational Linguistics.
- Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2022. [Finetuned language models are zero-shot learners](#). In *The Tenth International Conference on Learning Representa-*

- tions, *ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.
- Lingling Xu, Haoran Xie, Si-Zhao Joe Qin, Xiaohui Tao, and Fu Lee Wang. 2023. [Parameter-efficient fine-tuning methods for pretrained language models: A critical review and assessment](#). *CoRR*, abs/2312.12148.
- Adam X. Yang, Maxime Robeyns, Xi Wang, and Laurence Aitchison. 2024. [Bayesian low-rank adaptation for large language models](#). In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.
- Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. [Bitfit: Simple parameter-efficient fine-tuning for transformer-based masked language-models](#). In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), ACL 2022, Dublin, Ireland, May 22-27, 2022*, pages 1–9. Association for Computational Linguistics.
- Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. [Hellaswag: Can a machine really finish your sentence?](#) In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4791–4800. Association for Computational Linguistics.
- Jeffrey O. Zhang, Alexander Sax, Amir Zamir, Leonidas J. Guibas, and Jitendra Malik. 2020. [Side-tuning: A baseline for network adaptation via additive side networks](#). In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part III*, volume 12348 of *Lecture Notes in Computer Science*, pages 698–714. Springer.
- Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. 2023a. [LoRA-FA: Memory-efficient low-rank adaptation for large language models fine-tuning](#). *CoRR*, abs/2308.03303.
- Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. 2023b. [Adaptive budget allocation for parameter-efficient fine-tuning](#). In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.
- Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022a. [OPT: open pre-trained transformer language models](#). *CoRR*, abs/2205.01068.
- Yuanhan Zhang, Kaiyang Zhou, and Ziwei Liu. 2022b. [Neural prompt search](#). *CoRR*, abs/2206.04673.
- Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. 2024. [GaLore: Memory-efficient LLM training by gradient low-rank projection](#). In *Forty-first International Conference on Machine Learning, ICML 2024*.
- Han Zhou, Xingchen Wan, Ivan Vulic, and Anna Korhonen. 2024. [AutoPEFT: Automatic configuration search for parameter-efficient fine-tuning](#). *Trans. Assoc. Comput. Linguistics*, 12:525–542.
- Yaoming Zhu, Jiangtao Feng, Chengqi Zhao, Mingxuan Wang, and Lei Li. 2021. [Counter-interference adapter for multilingual machine translation](#). In *Findings of the Association for Computational Linguistics: EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 16-20 November, 2021*, pages 2812–2823. Association for Computational Linguistics.

## A Instruction Template

Regarding the instruction tuning of large LLMs, we prompt the model without step-wise reasoning using the Alpaca (Taori et al., 2023) prompt template presented below.

```
“Below is an instruction that describes a
task, paired with an input that provides
further context. Write a response that
appropriately completes the request.
### Instruction: {instruction}
### Input: {input}
### Response:
”
```

## B Software

Here we provide the details of the software used for the implementation of TOKENTUNE as well as the fine-tuning and evaluation of TOKENTUNE and baselines. Our implementation of TOKENTUNE builds upon the HuggingFace Transformers library<sup>6</sup> (v4.33.1). For LoRA (Hu et al., 2022), we used the HuggingFace PEFT library<sup>7</sup> (v.0.5.0). Datasets used for fine-tuning were obtained from the HuggingFace Datasets library<sup>8</sup> (v2.18.0). We used Open-Platypus<sup>9</sup> for fine-tuning. For the evaluation with the Llama2 model in Section 5, we used the lm-evaluation-harness framework<sup>10</sup> (v.0.4.2). We used the PyTorch framework<sup>11</sup> (v.2.0.1). Results from Table 1 are scored by the evaluation server.<sup>12</sup> As in Devlin et al. (2019), we discard results for the WNLI task.<sup>13</sup>

## C License

The majority of TOKENTUNE is licensed under CC-BY-NC, however portions of the project are available under separate license terms: Transformers is licensed under the Apache 2.0 license. The license of other libraries used for this paper is as follows. The PEFT and Datasets libraries from HuggingFace are under the Apache-2.0 license. The lm-evaluation-harness framework is under the MIT license. PyTorch is under the modified BSD-3

<sup>6</sup><https://github.com/huggingface/transformers>

<sup>7</sup><https://github.com/huggingface/peft>

<sup>8</sup><https://github.com/huggingface/datasets>

<sup>9</sup><https://huggingface.co/datasets/garage-bAInd/Open-Platypus>

<sup>10</sup><https://github.com/EleutherAI/lm-evaluation-harness>

<sup>11</sup><https://github.com/pytorch/pytorch>

<sup>12</sup><https://gluebenchmark.com/leaderboard>

<sup>13</sup>See (12) from <https://gluebenchmark.com/faq>

license. Open-Platypus used for fine-tuning consists of multiple datasets; their license information can be found at <https://huggingface.co/datasets/garage-bAInd/Open-Platypus>.

## D Training and Evaluation Data

BERT model has been pre-trained on 3,300M words. Regarding the instruction tuning experiments, we tuned the Llama2-7B on 21,221 samples from the Open-Platypus (Lee et al., 2023) dataset. Note that, while Open-Platypus consists of 11 open-source datasets, we exclude two of them<sup>14</sup> that include outputs from GPT (OpenAI, 2023), and instead use the other nine datasets for fine-tuning. Llama2-7B has been pre-trained on 2T tokens and fine-tuned on 100,000 samples.<sup>15</sup>

## E Memory Breakdown

Parameter-Efficient Fine-Tuning (PEFT) approaches aim at reducing the compute and storage requirements to fine-tune LLMs by only updating a small subset of the model parameters. As a result, we do not need to store any corresponding gradients and optimizer states for the frozen parameters. When parameters, gradients, and optimizer states represent the majority of the GPU memory usage, these PEFT methods can effectively reduce the memory cost. However, when most GPU memory is used to store intermediate activations, which are required for gradient computation during the backward pass, these PEFT methods cannot effectively cut down the memory cost.

Table 5 presents the GPU memory required to perform one training step with BERT-base (Devlin et al., 2019) and OPT (Zhang et al., 2022a) on a consumer hardware GPU. We calibrate the example such that the memory requirement is roughly the same for both models. In this configuration we can only fit a single example for OPT, while we can use a batch size of 256 for BERT. We observe that the memory breakdown is very different between the two configurations. The required memory drastically increases during the forward pass for BERT and during the backward pass for OPT. When comparing the execution of forward pass with and without enabling gradient computation in PyTorch, we estimate that the memory cost to store intermediate activations represents around 22 Gb for BERT

<sup>14</sup>leetcode-solutions-python-testgen-gpt4 and airoboros-gpt4-1.4.1

<sup>15</sup><https://llama.meta.com/llama2/>

Table 4: GPU memory required to fine-tune Llama2-7B (Touvron et al., 2023) using TOKENTUNE with a varying selection ratio, as well as QLoRA and LoRA. Since we could not perform full fine-tuning on our hardware, we estimate the full fine-tuning memory based on the memory reported for TOKENTUNE, TOKENTUNE + LoRA, and LoRA. See Section 5.3 and Figure 4 for details of the experiment.

Selection Ratio	TOKENTUNE (Ours) + QLoRA	QLoRA	TOKENTUNE (Ours) + LoRA	LoRA	TOKENTUNE (Ours)	Full Fine-Tuning
12.5%	11.7 GiB	51.9 GiB	44.6 GiB	80.4 GiB	64.0 GiB	91.4 GiB
25.0%	17.2 GiB	51.9 GiB	48.5 GiB	80.4 GiB	65.0 GiB	91.4 GiB
37.5%	22.0 GiB	51.9 GiB	53.7 GiB	80.4 GiB	66.3 GiB	91.4 GiB
50.0%	27.4 GiB	51.9 GiB	58.3 GiB	80.4 GiB	70.2 GiB	91.4 GiB
62.5%	32.7 GiB	51.9 GiB	63.0 GiB	80.4 GiB	74.6 GiB	91.4 GiB
75.0%	38.8 GiB	51.9 GiB	68.1 GiB	80.4 GiB	79.5 GiB	91.4 GiB
87.5%	43.7 GiB	51.9 GiB	73.4 GiB	80.4 GiB	83.8 GiB	91.4 GiB
99.9%	49.0 GiB	51.9 GiB	77.7 GiB	80.4 GiB	88.7 GiB	91.4 GiB

Table 5: Using two models requiring roughly the same GPU memory, we observe that the memory breakdown and the impact of PEFT methods application are very different. For each model, we show the evolution of the GPU memory ( $\times 10^3$  MiB) required for performing one training step for OPT-1B3 (Zhang et al., 2022a) with a batch size of 1 and a sequence length of 128 and BERT-base (Devlin et al., 2019) with a batch size of 256, a sequence length of 128. Fwd (w/o grad) corresponds to the execution of the forward pass, while disabling gradient computation.

	BERT	OPT	w/ LoRA	
			BERT	OPT
Cuda Context	0.8	0.8	0.8	0.8
+ Model weights	1.3	<b>5.8</b>	1.3	<b>5.8</b>
+ Fwd (w/o grad)	2.9	6.1	2.9	6.1
+ Fwd (w/ grad)	<b>24.8</b>	6.3	<b>20.6</b>	6.3
+ Bwd	25.2	<b>11.3</b>	21.0	6.3
+ Optimizer step	25.2	<b>21.4</b>	21.0	6.3

and less than 1 Gb for OPT. On the contrary, we estimate that computing and storing the parameter gradients increase the memory requirement by less than 1 Gb for BERT and around 5 Gb for OPT. When applying LoRA (Hu et al., 2022), a PEFT method, we observe that the memory drastically decreases for OPT, while having a less significant impact on BERT. These examples demonstrate that an effective memory reduction across different usage scenarios can be achieved by combining a suite of memory-efficient fine-tuning methods that can complement each other by reducing different parts of the memory footprint simultaneously.

## F MRPC and STS-B Descriptive Statistics

Table 6 describes the relation between the absolute and relative number of frozen input positions. The

Table 6: Distribution of the sentence length for the two GLUE subtasks (MRPC and STS-B).

Task	25th per- centile (P25%)	Avg. tokens per sentence	75th per- centile (P75%)	Max tokens per sentence	# Training Sen- tences
STS-B	19.0	27.8	31.0	125	5,749
MRPC	44.0	53.2	62.0	103	3,668

Table 7: Relative proportion of fine-tuned tokens averaged over MRPC and STS-B tasks with respect to the number of fine-tuned tokens, along with the corresponding average performance (reported in Figure 3 (right)).

# Fine-Tuned Tokens	Average Relative Proportion of Fine-Tuned Tokens	Average Perf.
4	13.6%	84.9
8	27.2%	86.4
16	53.9%	87.6
32	81.4%	88.4
64	99.0%	88.8

statistics include distribution of the sentence length for the two subtasks (MRPC and STS-B) used to produce Figure 3 (right). We also report in Table 7 the relative proportion of fine-tuned tokens averaged over MRPC and STS-B tasks, as the absolute number of fine-tuned tokens changes, along with the corresponding average performance, which is reported in Figure 3 (right).

## G GPU Memory Usage

Table 4 shows the GPU memory usage required to fine-tune Llama2-7B (Touvron et al., 2023) using the proposed TOKENTUNE with a varying selection ratio, as well as QLoRA and LoRA. Figure 4 also visualizes the same results. See Section 5.3 and Figure 4 for further details of the experiment.