DropBP: Accelerating Fine-Tuning of Large Language Models by Dropping Backward Propagation

Sunghyeon Woo[1†] **Baesung Park**[2*] **Byeongwook Kim**[2] **Minjung Jo**[2]
Se Jung Kwon[2] **Dongsuk Jeon**[1] **Dongsoo Lee**[2]
Seoul National University[1] NAVER Cloud[2]

Abstract

Large language models (LLMs) have achieved significant success across various domains. However, training these LLMs typically involves substantial memory and computational costs during both forward and backward propagation. While parameter-efficient fine-tuning (PEFT) considerably reduces the training memory associated with parameters, it does not address the significant computational costs and activation memory. In this paper, we propose Dropping Backward Propagation (DropBP), a novel approach designed to reduce computational costs and activation memory while maintaining accuracy. DropBP randomly drops layers during backward propagation, which is essentially equivalent to training shallow submodules generated by undropped layers and residual connections. Additionally, DropBP calculates the sensitivity of each layer to assign an appropriate drop rate, thereby stabilizing the training process. DropBP is not only applicable to full fine-tuning but can also be orthogonally integrated with all types of PEFT by dropping layers during backward propagation. Specifically, DropBP can reduce training time by 44% with comparable accuracy to the baseline, accelerate convergence to the same perplexity by 1.5×, and enable training with a sequence length 6.2× larger on a single NVIDIA-A100 GPU. Furthermore, our DropBP enabled a throughput increase of 79% on a NVIDIA A100 GPU and 117% on an Intel Gaudi2 HPU. The code is available at https://github.com/WooSunghyeon/dropbp.

1 Introduction

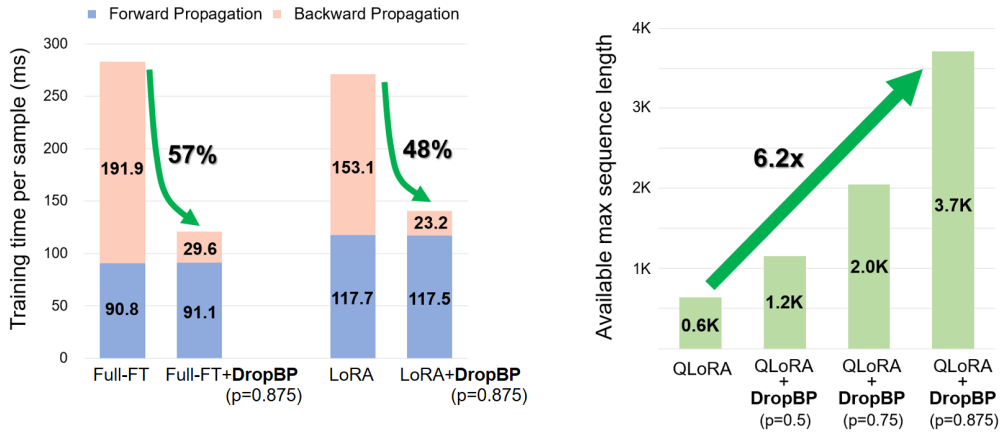
Since the advent of the transformer architecture [1], the field of language modelling has experienced dramatic advancements. Especially, following the scaling laws [2, 3], the development of Large Language Models (LLMs) [4, 5, 6, 7, 8, 9] continues with the aim of achieving or outperforming human capabilities. However, tremendously increasing the size of the model results in significant costs for training from scratch. An alternative approach for developing high-capability LLMs without the costly pretraining on extensive datasets is instruction tuning [10, 11, 12, 13]. This method fine-tunes well-trained foundation models on relatively small instruction-following datasets, enabling the models to better understand and follow prompts.

While fine-tuning Large Language Models (LLMs) on instruction-following datasets is more cost-effective than training from scratch, it still requires substantial memory for parameters and activations, along with significant floating-point operations (FLOPs). In this context, Parameter-Efficient Fine-Tuning (PEFT) techniques [14, 15, 16] effectively reduce the memory required for parameter gradients and optimizer states by freezing pretrained weights and selectively training newly added modules. Moreover, when combined with quantization methods [17, 18, 19, 20], these techniques can further significantly decrease the memory requirements for parameters.

[1]Equal contribution
[2]Intern at NAVER Cloud

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

 https://doi.org/10.52202/079017-0637



(a) Training time per sample of LLaMA2-7B. (b) Available max sequence length of LLaMA2-70B.

Figure 1: Performance enhancements in fine-tuning large language models using DropBP when the p represents the target average drop rate for backward propagation: (a) Training time per sample for fine-tuning LLaMA2-7B with DropBP, at a sequence length of 512 and a micro batch size of 2. (b) Available max sequence length for fine-tuning LLaMA2-70B with DropBP, at a micro batch size of 1 on an NVIDIA-A100 GPU.

While Parameter-Efficient Fine-Tuning (PEFT) has successfully reduced memory associated with parameters, two significant challenges remain for efficient fine-tuning: computational cost and activation memory, both of which are linked to the backpropagation [21]. First, fine-tuning Large Language Models (LLMs) using a backpropagation requires substantial floating-point operations (FLOPs). Specifically, the backpropagation algorithm necessitates forward propagation to calculate outputs and backward propagation to compute gradients for inputs and parameters. Notably, backward propagation demands twice the computational operations compared to forward propagation, thus becoming the primary bottleneck. Second, all intermediate outputs (i.e., activations) generated during forward propagation must be stored for compute in backward propagation. This activations consume considerable memory, which becomes especially critical when training LLMs on long sequence contexts [22, 23].

In this paper, we introduce Dropping Backward Propagation (DropBP), an efficient fine-tuning algorithm for LLMs that significantly reduces computational costs and activation memory. DropBP randomly drops layers during backward propagation, which is essentially equivalent to training shallow submodules generated by undropped layers and residual connections. As a result, these undropped layers no longer require FLOPs and activation memory during backward propagation. Additionally, DropBP calculates the sensitivity of each layer, an indicator of its impact on the total training process, to adjust drop rate. This careful calibration of drop rate according to layer sensitivity ensures more stable training. This DropBP algorithm can be seamlessly integrated with any PEFT algorithm, operating orthogonally by simply dropping layers during backward propagation.

We implemented DropBP as an easy-to-integrate PyTorch library [24], requiring only minimal changes to the existing training codes. In experiments, DropBP successfully reduces training time as shown in Fig. 1a, maintaining comparable accuracy on the MMLU [25] and commonsense reasoning tasks [26, 27, 28, 29, 30]. The DropBP also accelerated the convergence of the same perplexity by 1.5$\times$ in LLaMA2-70B [8]. Moreover, DropBP substantially decreases activation memory, increasing an available maximum sequence length by up to 6.2 $\times$ in LLaMA2-70B on a single NVIDIA A100 GPU [31], as shown in Fig. 1b. Finally, our DropBP increases training throughput by up to 79% and 117% on a single NVIDIA A100 GPU and Intel Gaudi2 HPU [32], respectively, when fully fine-tuning LLaMA3-8B [9]. In summary, the main contributions of our paper are:

- We propose DropBP, an efficient fine-tuning algorithm that randomly drops backward propagation based on layer sensitivity.

https://doi.org/10.52202/079017-0637

- We implemented DropBP as a user-friendly PyTorch extension with a straightforward API for ease of use, making it easily applicable to existing training codes.
- DropBP reduces training time by 44% with comparable accuracy, increases convergence speed by 1.5× , increases the available maximum sequence length up to 6.2×, and enhances training throughput up to 117%.

2 Background & Motivation

Backpropagation Backpropagation [33], a core algorithm for training deep neural networks, involves both forward and backward propagation. Specifically, the training process in the linear layer is represented as follows:

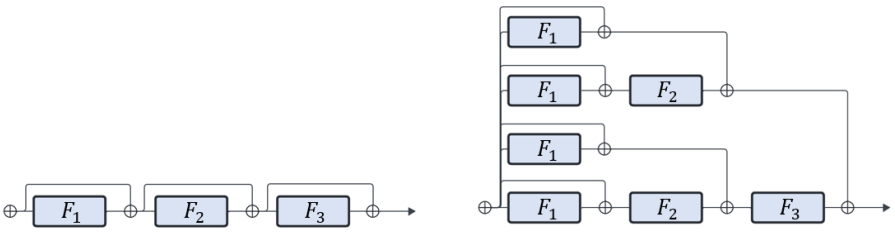
$$\text{Forward Prop:} \quad \mathbf{H}_{out} = \mathbf{W} \times \mathbf{H}_{in} \tag{1}$$

$$\text{Backward Prop:} \quad \nabla\mathbf{H}_{in} = \mathbf{W}^T \times \nabla\mathbf{H}_{out} \tag{2}$$

$$\nabla\mathbf{W} = \nabla\mathbf{H}_{out} \times \mathbf{H}_{in}^T \tag{3}$$

where **H** and **W** represent the activations and parameters, respectively, with '×' indicating matrix multiplication operation. The gradients of **H** and **W** are denoted by $\nabla\mathbf{H}$ and $\nabla\mathbf{W}$. The computational costs during forward propagation primarily arises from matrix multiplication for computing output activations by Eq. 1. In backward propagation, the computational burden is primarily due to matrix multiplication for calculating input gradients by Eq. 2 and parameter gradients by Eq. 3. Note that the computational costs of these equations are almost equal. Consequently, the FLOPs required for backward propagation including Eqs. 2 and 3 are approximately 2× as large as the FLOPs needed for forward propagation by Eq. 1. Furthermore, the activations of all layers ($\mathbf{H}_{in}^T$) must be stored in memory for use in backward propagation computations in Eq. 3. Therefore, focusing on reducing the computations during backward propagation is crucial for decreasing both the overall computational costs and the activation memory.

Interpretation the model with residual connections Residual connections are one of the widely used methods to address the issue of vanishing gradients [34]. Transformer [1] also incorporates residual connections that bypass multi-head attention and feedforward networks. Networks utilizing these residual connections can be interpreted as ensembles of several submodules [35]. For example, if we expand the model with three residual connections as shown in Fig. 2a, it can be represented as a combination of eight submodules, as depicted in Fig. 2b From this perspective, a network with n layers can be interpreted as an ensemble of 2^n submodules [35].



(a) 3-layers model with residual connections. (b) Combination of multiple submodules.

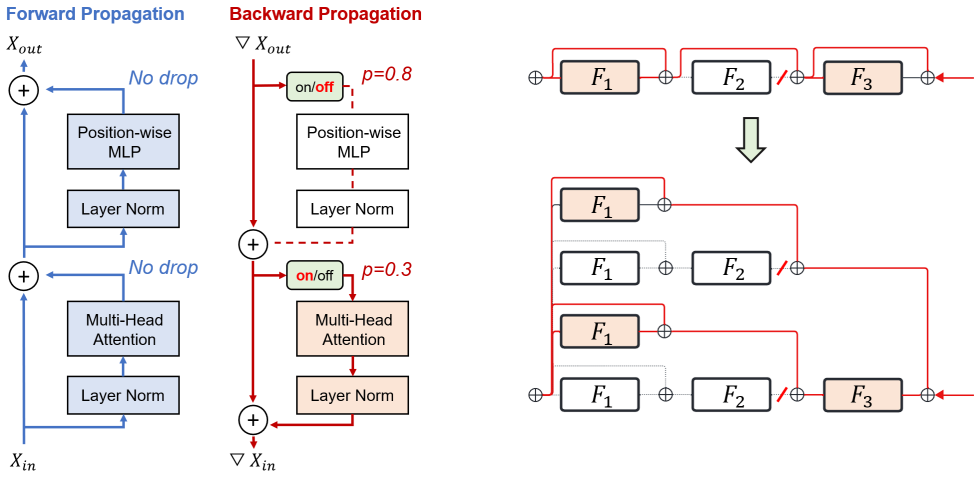
Figure 2: Interpreting the model with residual connections as a combination of multiple submodules.

3 Methodology

3.1 Dropping Backward propagation

In Section 2, we observed that the backpropagation algorithm consumes a significant amount of FLOPs and activation memory, particularly for the backward propagation. To reduce this overhead,

https://doi.org/10.52202/079017-0637



(a) The concept of DropBP. (b) DropBP as combiation of shallow submodules.

Figure 3: The overveiw of DropBP.

we propose a straightforward approach: Dropping Backward Propagation (DropBP). DropBP is designed to drop layers exclusively during backward propagation, effectively reducing both FLOPs and activation memory for the dropped layers, as demonstrated in following equations:

$$\mathbf{X}_{imm} = \mathbf{X}_{in} + \mathcal{D}_{p_i}(f_{ATTN}(f_{LN}(\mathbf{X}_{in}))) \tag{4}$$

$$\mathbf{X}_{out} = \mathbf{X}_{imm} + \mathcal{D}_{p_{i+1}}(f_{FFN}(f_{LN}(\mathbf{X}_{imm}))) \tag{5}$$

Here, $\mathbf{X}_{in}$, $\mathbf{X}_{out}$, and $\mathbf{X}_{imm}$ represent the input, output, and immediate activation between the attention layer and feedforward network in a transformer block, respectively. f_{ATTN}, f_{FFN}, and f_{LN} denote the attention layer, feedforward network, and layer normalization of the transformer block. The DropBP layer, defined as $\mathcal{D}_p(\mathbf{X})$, skips backward propagation in the input $\mathbf{X}$ with a probability of p, while not dropping forward propagation. Following to Eqs. 4 and 5, backward propagation in the attention layer and feedforward network is dropped with probabilities p_i and p_{i+1}, respectively, as shown in Fig. 3a.

We can view the transformer as a collection of a lot of submodules composed of various modules (i.e., $f_{ATTN} \circ f_{LN}$ and $f_{FFN} \circ f_{LN}$) with residual connections, as described in Section 2. When the transformer block contains n units, each including multi-head attention and a feedforward network, the model can be interpreted as an ensemble of 2^{2n} submodules. From this perspective, DropBP can be interpreted as training only certain shallow submodules. For example, as shown in Fig. 3b, if the F_2 layer is dropped, only the shallow submodule composed of the remaining layers is trained during backward propagation. Therefore, if the overall drop rate is p, DropBP can be interpreted as training shallow submodules with the depth of $2n(1-p)$ or less, since $2np$ layers are dropped. We anticipate that training these smaller modules alone can effectively fine-tune well-pretrained LLMs, based on the analysis that shallow submodules have a significant impact on the overall training process as detailed in Appendix A.

3.2 Sensitivity-based Drop Rate Allocation

In Section 3.1, we introduce DropBP, which selectively drops layers during backward propagation and trains only certain shallow submodules. In addition, we hypothesized that the significance of individual layers and the submodules encompassing these layers varies in their impact on the overall training process. Therefore, we assign different drop rate to each layer based on sensitivity, which is defined by defined by how much each layer and its encompassing submodules affect the overall training process in terms of parameter gradient. Specifically, we calculate the sensitivity of a layer by the variance in L2-norm of parameter gradients between when the backward propagation of that layer

https://doi.org/10.52202/079017-0637

is skipped or not, inspired by GradNormVar [36], a memory efficient gradient variance approximation, as below:

$$S_l = \sum_i (||\nabla \mathbf{W}_i||_2 - ||\nabla \mathbf{W}_i^{(l)}||_2)^2 \tag{6}$$

where S_l denotes of the l-th layer. Here, $\nabla \mathbf{W}_i$ represnets the parameter gradient of the i-th layer when no layers are dropped, while $\nabla \mathbf{W}_i^{(l)}$ denotes the parameter gradient of the i-th layer when the l-th layer is dropped during backward propagation. After calculating the sensitivity for each layer, we aim to minimize the expected sensitivities across all layers-essentially the expected gradient variance caused by DropBP-under a given FLOPs as follow:

$$\min \sum_i (1 - p_i) S_i, \quad \text{s.t.} \sum_i (1 - p_i) F_i \leq F_t \tag{7}$$

where p_i denotes the drop rate, and F_i denotes the FLOPs of the i-th layer during backward propagation. F_t represents the target FLOPs, derived from the target average drop rate p_{avg} (i.e. $F_t = (1 - p_{avg}) \sum_i F_i$). In other words, we determine the drop rates across all layers that minimize the expected sensitivities of the model, while satisfying a given FLOPs budget, by solving Problem 7. In practice, DropBP addresses the Prob. 7 using a simple greedy algorithm, as detailed in Section 4.1.

4 Evaluation

4.1 Implementation and Settings

```python
import torch
from dropbp.layer import DropBP

class Block(torch.nn.Modoule): # transformer block
    def __init__(self, ..):
        self.norm_1 = ...
        self.attn = ...
        self.norm_2 = ...
        self.mlp = ...
        self.dropbp_attn = DropBP(layers=[self.norm_1, self.attn], flops=..)
        self.dropbp_mlp = DropBP(layers=[self.norm_2, self.mlp], flops=..)
    def forward(self, x, ..):
        h = self.attn(self.norm_1(x), ...)
        x = self.dropbp_attn(h)+x   # instead of 'x = h+x'
        h = self.mlp(self.norm_2(x))
        x = self.dropbp_mlp(h)+x    # instead of 'x = h+x'
        return x
```

Step 1. Insert a DropBP layer into the Transformer Block

(a) Code for preparing a model with DropBP.

```python
import torch
from dropbp.handler import DropBPHandler

model = ... # define a model
optimizer = ... # define a optimizer
dropbp_handler = DropBPHandler(model)
dropbp_handler.set_initial_drop_rate(drop_rate)

# training loop
for iter in ...
    dropbp_handler.dropbp_handler.set_dropped_layers()
    def backprop:
        output = model(data)
        loss = loss_func(output, target)
        optimizer.zero_grad()
        loss.backward()
    if iter == int(max_iter * 0.1)
        dropbp_handler.sensitivity_based_drop_bp(backprop, drop_rate)

    out = model(data)
    loss = loss_func(output,target)
    non_grad = dropbp_handler.detact_non_grad()
    if not(non_grad):
        loss.backward()
    optimizer.step()
    ...
```

Step 2. Set the initial drop rate

Step 3. Set the dropped layers for each iteration

Step 4. Adjust the drop rates of layers based on sensitivities

Step 5. Exclude the case where all layers are dropped

(b) Code for training with DropBP.

Figure 4: Code implementation for integrating DropBP.

DropBP aims to decrease the training costs in fine-tuning based on backpropagation, consequently enabling the acceleration of both full fine-tuning and parameter-efficient fine-tuning using back-propagation. To facilitate practical implementation, we developed a user-friendly DropBP library in PyTorch [24], as demonstrated in Fig. 4. In detail, we implemented a `DropBP` layer that internally drops backward propagation in the input direction according to a given drop rate as shown in Fig. 4a. The `DropBP` layer designed to initially receive the FLOPs of the layers that would be skipped (F_i) as initial value to solve Prob. 7.

Additionally, we developed a `DropBPHandler` that automatically solve Prob. 7 by assigning varying drop rate to each layer using a simple greedy algorithm, as demonstrated in Fig. 4b. Specifically, the process begins by setting the drop rate (p_i) of all layers to 0 and then gradually increases them to

https://doi.org/10.52202/079017-0637

align with the target average drop rate (p_{avg}) set by the user. In each step, the drop rate for each layer is incremented by 0.1, ensuring that the increase in total expected sensitivity is kept to a minimum. We train with uniform drop rate for the initial 10% of total iterations and then adjust the drop rate for each layer based on sensitivity when training is stable. Since sensitivity is calculated only once at the 10% of the entire iteration, the overhead from this calculation is negligible.

We integrated our DropBP code into *LitGPT* [37] and *HuggingFace* [38], repositories for training and evaluating LLMs. We first fine-tuned LLaMA2-7B, 13B, and 70B [8] on Alpaca [11] and Dolly [13] datasets. The fine-tuned models were evaluated on 5-shot Massive Multitask Language Understanding (MMLU) tasks [25] and 0-shot commonsense reasoning tasks [26, 27, 28, 29, 30] using *lm-evaluation-harness* [39] library. We also fine-tunes LLaMA3-8B [9] on the Oasst1 dataset [40] and evaluating the model on MT-Bench task [41] using GPT4o-mini [5] as a judge. These experiments were conducted on a single NVIDIA A100 GPU, and we measured the training time, memory usage, and convergence speed. More details about setup can be found in Appendix F.

Table 1: Test accuracy on MMLU and commonsense reasoning tasks. In DropBP, drop rate is the target average drop rate across all layers in backward propagation. Note that Full-FT stands for full fine-tuning.

Method	Drop Rate	Dataset	MMLU (5-shot)					Commonsense Reasoning (0-shot)						
			Human.	STEM	Social.	Other	Avg.	PIQA	HS	Arc-C	Arc-E	OBQA	WG	Avg.
LLaMA2-7B	-	-	43.5	37.0	51.6	52.2	45.7	79.1	76.0	46.2	74.5	44.0	69.3	64.9
LoRA	-	Alpaca	42.7	36.3	50.0	51.2	44.7	80.0	75.9	48.5	75.0	46.2	70.5	66.0
LoRA+DropBP	**0.5**	Alpaca	42.8	35.7	50.3	51.0	**44.7**	79.6	76.0	48.7	75.5	46.2	69.6	65.9
LoRA+DropBP	0.75	Alpaca	41.4	36.3	48.4	50.6	43.8	79.5	76.9	48.0	75.6	47.4	69.0	66.1
LoRA+DropBP	**0.875**	Alpaca	41.5	34.5	49.6	50.4	43.7	79.6	77.2	48.2	76.3	47.8	69.1	**66.4**
Full-FT	-	Alpaca	42.7	35.6	50.4	51.1	44.7	79.2	76.1	48.0	75.8	45.2	69.8	65.7
Full-FT+DropBP	0.5	Alpaca	42.6	35.4	49.8	51.0	44.4	79.5	76.2	47.8	75.4	45.4	68.5	65.5
Full-FT+DropBP	**0.75**	Alpaca	42.6	36.7	51.2	50.9	**45.0**	78.8	77.0	48.6	75.8	45.6	69.8	**65.9**
Full-FT+DropBP	0.875	Alpaca	42.7	35.3	50.7	51.2	44.7	79.2	76.9	46.8	75.3	46.2	69.3	65.6
LoRA	-	Dolly	43.9	38.4	53.0	53.3	**46.7**	79.0	76.2	47.7	77.0	45.0	69.7	**65.8**
LoRA+DropBP	0.5	Dolly	44.0	36.8	53.1	53.2	46.4	79.3	76.3	47.3	76.5	44.8	68.8	65.5
LoRA+DropBP	0.75	Dolly	43.9	37.0	52.4	53.1	46.3	79.4	76.2	46.2	75.4	44.8	68.8	65.1
LoRA+DropBP	0.875	Dolly	43.6	36.7	52.3	53.0	46.1	79.1	76.1	45.8	75.3	44.6	68.4	64.9
Full-FT	-	Dolly	43.3	38.1	53.6	53.2	**46.6**	79.3	76.2	46.8	76.2	44.2	68.9	**65.3**
Full-FT+DropBP	0.5	Dolly	43.4	37.1	52.9	53.0	46.2	79.2	76.2	46.4	75.6	44.4	68.8	65.1
Full-FT+DropBP	0.75	Dolly	43.1	36.7	51.8	52.6	45.7	79.2	76.4	45.8	75.4	44.6	69.1	65.1
Full-FT+DropBP	0.875	Dolly	42.5	36.8	52.4	52.4	45.6	79.2	76.3	46.2	75.0	44.8	69.0	65.1
LLaMA2-13B	-	-	52.2	44.1	62.9	61.5	54.8	80.6	79.4	49.5	77.4	45.6	72.5	67.5
LoRA	-	Alpaca	51.7	43.8	63.3	61.7	54.7	80.6	79.5	51.6	78.5	45.8	72.1	68.0
LoRA+DropBP	0.5	Alpaca	52.4	44.2	63.1	62.0	55.0	80.7	79.6	50.9	78.4	44.8	71.7	67.7
LoRA+DropBP	**0.75**	Alpaca	52.1	44.2	64.1	61.6	**55.1**	81.0	79.7	51.5	79.1	45.6	71.7	**68.1**
LoRA+DropBP	0.875	Alpaca	51.1	44.2	63.3	61.6	54.6	80.8	79.8	51.0	78.2	45.0	71.4	67.7
LoRA	-	Dolly	51.9	43.6	63.7	62.0	54.8	80.4	79.9	51.1	78.4	45.6	71.7	67.9
LoRA+DropBP	**0.5**	Dolly	52.4	44.1	63.4	62.1	55.1	80.7	79.9	50.9	78.5	45.6	72.3	**68.0**
LoRA+DropBP	0.75	Dolly	52.1	44.3	63.3	61.7	54.9	80.6	79.8	51.4	77.8	45.4	72.1	67.8
LoRA+DropBP	**0.875**	Dolly	52.8	43.9	63.4	61.9	**55.1**	80.5	79.7	51.3	77.9	45.2	72.0	67.8
LLaMA2-70B	-	-	64.7	57.0	79.6	74.0	68.3	82.4	83.0	57.3	80.6	48.6	77.4	71.6
QLoRA	-	Alpaca	64.9	57.0	79.6	74.0	68.3	82.8	83.3	59.6	82.2	48.4	78.5	72.5
QLoRA+DropBP	0.5	Alpaca	65.8	56.2	78.8	73.0	68.1	83.2	82.9	60.2	82.2	48.0	77.9	72.4
QLoRA+DropBP	0.75	Alpaca	65.0	55.2	78.8	73.5	67.7	83.5	83.1	58.9	81.3	48.2	77.3	72.0
QLoRA+DropBP	**0.875**	Alpaca	66.4	56.3	79.9	74.1	**68.8**	83.4	83.7	60.1	81.6	48.6	78.0	**72.6**
QLoRA	-	Dolly	65.5	58.0	79.7	74.5	68.9	82.8	83.3	58.3	81.2	48.0	77.4	71.8
QLoRA+DropBP	0.5	Dolly	65.1	57.2	79.3	74.1	68.4	82.8	83.4	57.8	81.7	47.6	78.1	71.9
QLoRA+DropBP	**0.75**	Dolly	65.1	57.4	79.7	74.5	**68.7**	82.4	83.5	58.4	82.0	48.2	77.8	**72.0**
QLoRA+DropBP	0.875	Dolly	65.4	56.6	79.6	74.1	68.5	83.1	83.1	57.6	81.6	48.0	78.5	72.0

Table 2: Training time, memory usage, and test score on MT-Bench task when fine-tuning LLaMA3-8B with DropBP on Oasst1 datasets.

Method	Mem	Time	Human.	STEM	Role.	Extract.	Writing	Reason.	Coding	Math	Avg.
No-tunes	-	-	6.25	5.70	5.45	4.85	5.20	4.40	3.20	1.95	4.62
LoRA	57G	27m	7.00	6.40	5.70	5.80	5.30	4.55	3.25	2.95	**5.12**
+DropBP (p=0.5)	42G	21m	6.55	6.25	6.05	5.50	5.05	4.45	3.75	3.25	5.11
+DropBP (p=0.75)	36G	17m	6.75	5.90	5.80	5.70	5.35	4.30	3.60	3.30	5.09
+DropBP (p=0.875)	**32G**	**16m**	6.60	6.55	5.90	5.70	5.70	3.95	3.40	2.80	5.08

4.2 Main Results: Accuracy and Efficiency

Accuracy on MMLU and Commonsense Reasoning We employ DropBP to accelerate baseline fine-tuning processes, including full fine-tuning (Full-FT), LoRA [14], and QLoRA [18], on the

Alpaca and Dolly datasets. As demonstrated in Table 1, DropBP achieves accuracy comparable to the baseline, with deviations less than 1% in all scenarios, and it even outperforms the baseline in several instances. Specifically, when DropBP is applied to fine-tune LLaMA2-7B, there is a 1% or less decrease in 5-shot MMLU accuracy compared to the baseline, while maintaining comparable 0-shot commonsense reasoning accuracy. In contrast, for LLaMA2-13B and LLaMA2-70B, fine-tuning with DropBP results in almost no decrease in accuracy on the MMLU and commonsense reasoning tasks, even at the high drop rate of 0.875.

Accuracy on MT-Bench Similar trends are observed in the MT-Bench tasks, with negligible decreases in accuracy as shown in Table 2. Specifically, when fine-tuning LLaMA3-8B on the Oasst1 dataset, DropBP generates responses of comparable quality to the baseline across various generation tasks. Although scores slightly decrease as the DropBP rate increases, the model fine-tuned with a high DropBP rate of 0.875 still achieves significantly higher scores compared to non-tuned models.

Table 3: Time required for fine-tuning LLaMA2 models with DropBP on the Alpaca datasets when p denotes the target average drop rate. The number of fine-tuning samples is 50K.

Model	Precision	PEFT	DropBP			
			p=0 (Baseline)	p=0.5	p=0.75	p=0.875
LLaMA2-7B	BF16-mixed	LoRA	2.2h	1.7h	1.4h	**1.3h**
	BF16	Full-FT	2.0h	1.3h	1.0h	**0.8h**
LLaMA2-13B	BF16	LoRA	2.9h	2.1h	1.7h	**1.5h**
LLaMA2-70B	BF16	QLoRA	29.6h	22.2h	18.4h	**16.5h**

Training Speed and Memory Usage We measured the fine-tuning time required to obtain the results in Table 1, as presented in Table 3. When using DropBP to LoRA or QLoRA, training time is reduced by 25%, 38%, and 44% at drop rates of 0.5, 0.75, and 0.875, respectively. In contrast, using DropBP to Full-FT resulted in even higher training time reductions of 33%, 50%, and 57% at the same drop rates. These findings align with the theoretical reduction in FLOPs due to DropBP, as detailed in Appendix B. We also confirmed that DropBP can significantly reduce memory usage during fine-tuning, as shown in Table 2. Specifically, while not using DropBP results in a memory consumption of 57GB, applying DropBP with a drop rate of 0.875 reduces memory usage to 32GB by eliminating the storage of activation memory for dropped layers. Additionally, we evaluated the convergence speed to reach the same validation perplexity (PPL) on downstream tasks, as illustrated in Fig. 5 and Fig. 10-11 in Appendix C. The results indicate that our DropBP increases training speed by up to 1.5× compared to the baseline in LLaMA2-70B.

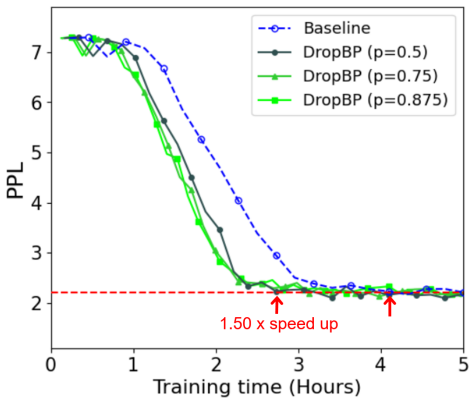


Figure 5: Validation perplexity (PPL) for fine-tuning LLaMA2-70B through QLoRA (baseline) with DropBP on the Alpaca dataset. The p represents the target average drop rate for backward propagation.

4.3 Usability of DropBP

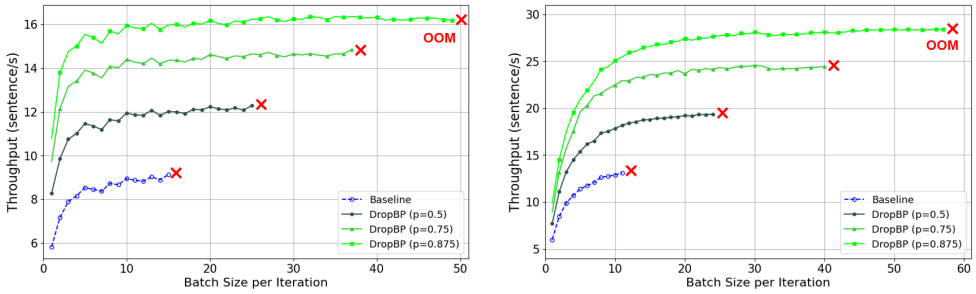
In this section, we evaluate the usability of DropBP, including its ability to train on long sequence data and its training throughput in constrained environments, such as a single NVIDIA A100 GPU or Intel Gaudi2 HPU.

Table 4: Available maximum sequence length for fine-tuning LLaMA2-70B using QLoRA with DropBP on a NVIDIA A100 GPU, at a micro batch size of 1.

Method	QLoRA	w/ DropBP		
Drop Rate	-	0.5	0.75	**0.875**
Max Seq Len	0.6K	1.2K (2.0×)	2.0K (3.3×)	**3.7K** (**6.2**×)

https://doi.org/10.52202/079017-0637

We first measured the maximum sequence length that could be trained without an Out Of Memory (OOM) on a single NVIDIA A100 GPU. The results in Table 4 indicate that our DropBP considerably increases the maximum sequence length, by up to 6.2× the baseline when the drop rate was 0.875. This is because DropBP allows skipping certain layers during backward propagation, eliminating the need to store activations required for calculating parameter gradients of those skipped layers. We believe that this property of DropBP will be particularly useful for fine-tuning LLMs with long-context data [22, 23].



(a) Throughput in a single A100 GPU.

(b) Throughput in a single Gaudi2 HPU.

Figure 6: Throguhput (sentences/s) on a single NVIDIA A100 GPU on a single NVIDIA A100 GPU and Intel Gaudi2 HPU when fine-tuning LLaMA3-8B with a sequence length of 512.

We also evaluated training throughput when full fine-tuning LLaMA3-8B using BF16 precision on a single NVIDIA A100 GPU and an Intel Gaudi2 HPU, increasing the batch size up to the point of OOM errors. As shown in Fig. 6, applying DropBP allows for an increase in batch size per iteration by up to 3.3× on the NVIDIA A100 GPU and 5.2× on the Intel Gaudi2 HPU, ensuring high hardware utilization and scalability. Furthermore, DropBP demonstrates a sustained increase in throughput over the baseline at an identical batch size. Ultimately, with a drop rate of 0.875, DropBP achieves a throughput of 16.4 sentences/s on the NVIDIA A100 GPU and 28.4 sentences/s on the Intel Gaudi2 HPU, increasing by 79% and 117% over the baseline, respectively.

4.4 Ablation Study

Impact of the Number of Submodules We conducted an ablation study to investigate the impact of the number of trainable submodules on the fine-tuning of LLMs. This study compared DropBP, which trains varying submodules randomly at each iteration, with layer freezing, which trains submodules composed of only upper layers. Here, the skip rate p denotes the drop rate in DropBP and the proportion of layers that are frozen in the layer freezing.

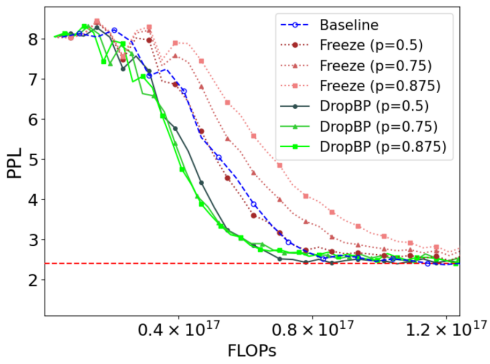


Figure 7: Validation perplexity (PPL) for fine-tuning LLaMA2-7B through LoRA with layer freezing or DropBP on the Alpaca dataset.

First, we analyzed the number of submodules trained by layer freezing and DropBP. In the case of layer freezing, the lower $2np$ layers are frozen and only the remaining $2n(1-p)$ layers are trained. In this case, the number of trainable upper submodules is $2^{2n(1-p)}$. In contrast, DropBP randomly drops $2np$ layers at each iteration, allowing it to train all submodules with a depth of $2n(1-p)$ or less without the restriction of training only the submodules composed of the upper layers. In this scenario, since the number of different submodules at depth i in the entire network is ${}_{2n}C_i$, DropBP can train $\sum_{i=0}^{2n(1-p)} {}_{2n}C_i$ submodules.

As shown in Table 5, when fine-tuning LLaMA2-7B using layer freezing or DropBP with a high skip rate of 0.875, we observed a significant 1.8% decrease in accuracy with layer freezing compared to

the baseline, while DropBP exhibited a relatively smaller accuracy decrease of 1.0%. Furthermore, as illustrated in Fig. 7, the convergence speed to the same validation PPL on the downstream task is much slower for layer freezing compared to DropBP, especially at high skip rates, where it converges even more slowly than the baseline. We believe this is due to the ability of DropBP to train a relatively larger number of submodules ($\sum_{i=0}^{8} {}_{64}C_i$), compared to the fewer submodules trained by layer freezing (2^8). Moreover, when fine-tuning LLaMA2-70B, DropBP resulted in a 0.5% increase in MMLU 5-shot accuracy compared to the baseline, despite a high skip rate of 0.875. This improvement is due to the large number of layers in LLaMA2-70B, enabling DropBP to train deeper and more numerous submodules ($\sum_{i=0}^{20} {}_{160}C_i$) even with a high skip rate of 0.875.

Table 5: The number of submodules being trained and test accuracy on the 5-shot MMLU tasks with layer freezing or DropBP on the Alpaca datasets.

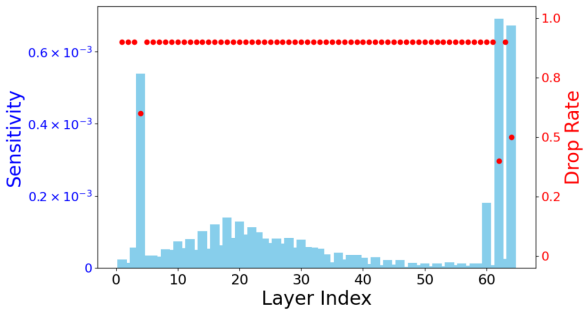
	LLaMA2-7B			LLaMA2-70B	
Method	**LoRA**	**LoRA+Freeze**	**LoRA+DropBP**	**QLoRA**	**QLoRA+DropBP**
Drop Rate	-	0.875	0.875	-	0.875
# of Submodules	2^{64}	2^8	$\sum_{i=0}^{8} {}_{64}C_i$	2^{160}	$\sum_{i=0}^{20} {}_{160}C_i$
Accuracy (%)	44.7	42.9 (-1.8)	43.7 (-1.0)	68.3	68.8 (+0.5)

Impact of Sensitivity-based Drop Rate We also conducted an ablation study to analyze the effectiveness of sensitivity-based drop rate allocations. First, we identified the sensitivity of different layers by calculating Eq. 6 during the training of LLMs in various scenarios, as illustrated in Fig. 8a and Fig. 7 in Appendix D. While the distribution varies slightly depending on the number of parameters, fine-tuning approach, and target average drop rate, there is a consistent tendency to assign importance to both the initial and final layers. Consequently, drop rates for these layers are allocated to be lower by a simple greedy algorithm, as explained in Section 4.1

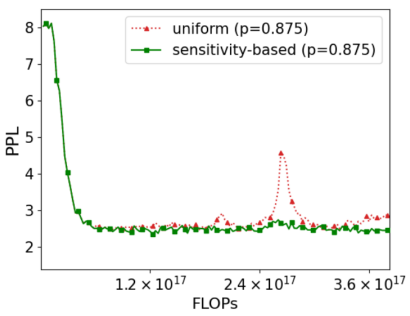
Table 6: Test accuracy on the 0-shot commonsense reasoning tasks when fine-tuning LLaMA2-7B and 13B through LoRA with DropBP at uniform or sensitivity-based drop rate on the Alpaca datasets. The target average drop rate is 0.875.

LLaMA2	**7B**		**13B**	
LR	**1e-4**	**3e-4**	**1e-4**	**3e-4**
LoRA	65.7	66.0	68.2	68.0
+DropBP (uniform)	66.4	63.1	66.6	65.8
+DropBP (sens)	66.6	64.7	67.7	67.3

Additionally, we fine-tuned the LLaMA2-7B and 13B using DropBP on Alpaca datasets, comparing sensitivity-based allocated drop rates with uniform drop rate. In detail, we compared the average accuracy of commonsense reasoning tasks when fine-tuning the models with a learning rate of 1e-4 and 3e-4, as shown in Table 6. Note that the PPL for fine-tuning LLaMA2-7B in Fig. 8b corresponds to a learning rate of 3e-4. The results indicate that sensitivity-based drop rates achieved a 1.6% higher



(a) Distribution of drop rates determined by sensitivity when the average drop rate is set to 0.875.



(b) Validation PPL with uniform and sensitivity-based allocated drop rates.

Figure 8: Distribution of drop rates and the validation PPL when fine-tuning LLaMA2-7B through LoRA with DropBP at uniform or sensitivity-based drop rate on Alpaca datasets.

https://doi.org/10.52202/079017-0637

accuracy compared to uniform drop rates with a relatively high learning rate of 3e-4, while there was no significant difference in accuracy when the learning rate was set to 1e-4 in LLaMA2-7B. Fig. 8b also shows that sensitivity-based drop rates consistently stabilized the convergence of validation loss, whereas uniform drop rates occasionally diverged when the learning rate was set to 3e-4 in LLaMA2-7B. This phenomenon is even more pronounced with LLaMA2-13B, resulting in a 1.1% increase in accuracy through sensitivity-based drop rate allocation, even with a low learning rate of 1e-4. In other words, sensitivity-based drop rate allocation helps stabilize the training process, especially in the case of large learning rates or larger models.

5 Related Works

Parameter-efficient fine-tuning When fine-tuning LLM, substantial amount of memory is required to store parameters, gradients, and optimizer states. LoRA [14] successfully reduces the memory allocated to gradients and optimizer states by inserting trainable rank decomposition matrices into the linear layers of the model while keeping the original LLM parameters frozen. LLaMA-Adapter [15] and LLaMA-Adapter V2 [16] significantly reduce training memory using trainable adoption prompts and zero-initialized attention mechanisms. Some studies attempt to reduce not only the memory footprint of gradients and optimizer states but also that of parameters by considering quantization. PEQA [20], for instance, quantizes the original LLM parameters into a low-bit format and fine-tunes only the scale factor, thus saving memory for parameters during training. QLoRA [18] and QA-LoRA [19], built upon LoRA, also employ quantization on the original LM parameters, significantly reducing parameter memory during training. Our DropBP is orthogonal and easily combinable with these PEFT techniques, enabling memory and computationally efficient fine-tuning.

Parallelism Parallelism techniques are widely used to accelerate training LLM using multiple GPU efficiently. Data parallelism [42] is a technique that involves dividing data along the batch dimension for training across multiple GPUs, which still requires sufficient memory to load the entire model on each GPU. Conversely, tensor parallelism [43, 44, 45] partitions the model across GPUs, dividing matrix multiplication operations column-wise and row-wise. Pipeline parallelism [46, 47, 48] involves partitioning the model depth-wise across GPUs, which enables efficient pipeline scheduling. The Zero Redundancy Optimizer (ZeRO) [49] and Fully Sharded Data Parallelism (FSDP) [50] shard parameters, gradients, and optimizer states across multiple GPUs, retrieving parameters when needed to restore their non-partitioned form, enabling the overlapping of computation and communication during training. While these parallelism techniques are designed to efficiently manage the massive computational costs across multiple GPUs, our DropBP specifically aims to reduce the inherent computational costs required for training process.

Layer dropping Stochastic Depth [51], the first approach to randomly drop layers during neural network training, reduces overfitting and costs in image recognition. Layerdrop [52] randomly drops layers during training and selectively uses some layers during inference, accelerating both processes for transformers. Progressive Layer Dropping (PLD) [53] progressively increases the drop rate across depth and iterations, improving training speed without accuracy degradation in transformers. These techniques speed up pretraining of small transformer models like BERT [54] by dropping layers during the entire training process, whereas DropBP, specific to fine-tuning LLMs, drops layers only during backward propagation. Consequently, as detailed in Appendix E, our DropBP achieves higher performance compared to these layer dropping when fine-tuning LLMs.

6 Conclusion

We propose DropBP, an effective algorithm that accelerates the fine-tuning of LLMs by randomly dropping layers during backward propagation, which can be orthogonally integrated into both full-fine tuning and parameter-efficient fine-tuning. We developed the DropBP library as a user-friendly PyTorch extension to facilitate easy integration into existing training codes. Experimental results demonstrate that DropBP significantly accelerates training speed during the fine-tuning of LLMs, achieving comparable accuracy to baseline fine-tuning. Furthermore, DropBP reduces activation memory, enabling long-context training and increasing batch size on limited resources. Consequently, applying DropBP enables a 79% higher throughput on an NVIDIA A100 GPU and a 117% higher throughput on an Intel Gaudi2 HPU.

Acknowledgment

This research was supported in part by the NAVER-Intel Co-Lab. The work was conducted by Seoul National University and reviewed by both NAVER and Intel.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Neurips, Long Beach, CA, USA, December 4-9, 2017,*, pages 5998–6008, 2017.

[2] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *CoRR*, abs/2001.08361, 2020.

[3] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katherine Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. An empirical analysis of compute-optimal large language model training. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh, editors, *NeurIPS, New Orleans, LA, USA November 28 - December 9, 2022*, 2022.

[4] 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. Language models are few-shot learners. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS, virtual, December 6-12, 2020*, 2020.

[5] OpenAI. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023.

[6] Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy P. Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul Ronald Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, and et al. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805, 2023.

[7] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971, 2023.

[8] 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

 https://doi.org/10.52202/079017-0637

Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288, 2023.

[9] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. The llama 3 herd of models. *CoRR*, abs/2407.21783, 2024.

[10] Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *ICLR, virtual, April 25-29, 2022*. OpenReview.net, 2022.

[11] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[12] Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. LIMA: less is more for alignment. *CoRR*, abs/2305.11206, 2023.

[13] Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. Free dolly: Introducing the world's first truly open instruction-tuned llm. Technical report, Databricks, 2023.

[14] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. In *ICLR, April 25-29, 2022, virtual*. OpenReview.net, 2022.

[15] Renrui Zhang, Jiaming Han, Aojun Zhou, Xiangfei Hu, Shilin Yan, Pan Lu, Hongsheng Li, Peng Gao, and Yu Qiao. Llama-adapter: Efficient fine-tuning of language models with zero-init attention. *CoRR*, abs/2303.16199, 2023.

[16] Peng Gao, Jiaming Han, Renrui Zhang, Ziyi Lin, Shijie Geng, Aojun Zhou, Wei Zhang, Pan Lu, Conghui He, Xiangyu Yue, Hongsheng Li, and Yu Qiao. Llama-adapter V2: parameter-efficient visual instruction model. *CoRR*, abs/2304.15010, 2023.

[17] Se Jung Kwon, Jeonghoon Kim, Jeongin Bae, Kang Min Yoo, Jin-Hwa Kim, Baeseong Park, Byeongwook Kim, Jung-Woo Ha, Nako Sung, and Dongsoo Lee. Alphatuning: Quantization-aware parameter-efficient adaptation of large-scale pre-trained language models. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang, editors, *EMNLP, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 3288–3305. Association for Computational Linguistics, 2022.

[18] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *CoRR*, abs/2305.14314, 2023.

[19] Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. Qa-lora: Quantization-aware low-rank adaptation of large language models. *CoRR*, abs/2309.14717, 2023.

[20] Jeonghoon Kim, Jung Hyun Lee, Sungdong Kim, Joonsuk Park, Kang Min Yoo, Se Jung Kwon, and Dongsoo Lee. Memory-efficient fine-tuning of compressed large language models via sub-4-bit integer quantization. *CoRR*, abs/2305.14152, 2023.

[21] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[22] Yukang Chen, Shengju Qian, Haotian Tang, Xin Lai, Zhijian Liu, Song Han, and Jiaya Jia. Longlora: Efficient fine-tuning of long-context large language models. *CoRR*, abs/2309.12307, 2023.

[23] Wenhan Xiong, Jingyu Liu, Igor Molybog, Hejia Zhang, Prajjwal Bhargava, Rui Hou, Louis Martin, Rashi Rungta, Karthik Abinav Sankararaman, Barlas Oguz, Madian Khabsa, Han Fang, Yashar Mehdad, Sharan Narang, Kshitiz Malik, Angela Fan, Shruti Bhosale, Sergey Edunov, Mike Lewis, Sinong Wang, and Hao Ma. Effective long-context scaling of foundation models. *CoRR*, abs/2309.16039, 2023.

[24] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *NeurIPS 2019, Vancouver, BC, Canada, December 8-14, 2019*, pages 8024–8035, 2019.

[25] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *ICLR, virtual, Austria, May 3-7, 2021*. OpenReview.net, 2021.

[26] Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: reasoning about physical commonsense in natural language. In *AAI, New York, NY, USA, February 7-12, 2020*, pages 7432–7439. AAAI Press, 2020.

[27] Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. Hellaswag: Can a machine really finish your sentence? In Anna Korhonen, David R. Traum, and Lluís Màrquez, editors, *ACL, Florence, Italy, July 28- August 2, 2019*, pages 4791–4800. Association for Computational Linguistics, 2019.

[28] Sumithra Bhakthavatsalam, Daniel Khashabi, Tushar Khot, Bhavana Dalvi Mishra, Kyle Richardson, Ashish Sabharwal, Carissa Schoenick, Oyvind Tafjord, and Peter Clark. Think you have solved direct-answer question answering? try arc-da, the direct-answer AI2 reasoning challenge. *CoRR*, abs/2102.03315, 2021.

[29] Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? A new dataset for open book question answering. In Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii, editors, *EMNLP, Brussels, Belgium, October 31 - November 4, 2018*, pages 2381–2391. Association for Computational Linguistics, 2018.

[30] Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. Winogrande: An adversarial winograd schema challenge at scale. In *AAAI, New York, NY, USA, February 7-12, 2020*, pages 8732–8740. AAAI Press, 2020.

[31] Martin Svedin, Steven W. D. Chien, Gibson Chikafa, Niclas Jansson, and Artur Podobas. Benchmarking the nvidia gpu lineage: From early k80 to modern a100 with asynchronous memory transfers. *arXiv preprint arXiv:2106.04979*, 2021.

[32] Intel Corporation. Intel gaudi2 ai accelerators white paper. Technical report, Intel Corporation, 2023.

[33] Henry J Kelley. Gradient theory of optimal flight paths. *Ars Journal*, 30(10):947–954, 1960.

[34] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 770–778. IEEE Computer Society, 2016.

[35] Andreas Veit, Michael J. Wilber, and Serge J. Belongie. Residual networks behave like ensembles of relatively shallow networks. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 550–558, 2016.

[36] Sunghyeon Woo, Sunwoo Lee, and Dongsuk Jeon. ALAM: Averaged low-precision activation for memory-efficient training of transformer models. In *The Twelfth International Conference on Learning Representations*, 2024.

 https://doi.org/10.52202/079017-0637

[37] Lightning-AI. Lit-gpt. `https://github.com/Lightning-AI/lit-gpt`, 2023.

[38] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771, 2019.

[39] Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. A framework for few-shot language model evaluation, 12 2023.

[40] Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. Openassistant conversations - democratizing large language model alignment. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *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*, 2023.

[41] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *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*, 2023.

[42] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training. *Proc. VLDB Endow.*, 13(12):3005–3018, 2020.

[43] Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *CoRR*, abs/1909.08053, 2019.

[44] Shaden Smith, Mostofa Patwary, Brandon Norick, Patrick LeGresley, Samyam Rajbhandari, Jared Casper, Zhun Liu, Shrimai Prabhumoye, George Zerveas, Vijay Korthikanti, Elton Zheng, Rewon Child, Reza Yazdani Aminabadi, Julie Bernauer, Xia Song, Mohammad Shoeybi, Yuxiong He, Michael Houston, Saurabh Tiwary, and Bryan Catanzaro. Using deepspeed and megatron to train megatron-turing NLG 530b, A large-scale generative language model. *CoRR*, abs/2201.11990, 2022.

[45] Vijay Korthikanti, Jared Casper, Sangkug Lym, Lawrence McAfee, Michael Andersch, Mohammad Shoeybi, and Bryan Catanzaro. Reducing activation recomputation in large transformer models. *CoRR*, abs/2205.05198, 2022.

[46] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Xu Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett, editors, *NeurIPS 2019, Vancouver, BC, Canada, VDecember 8-14, 2019*, pages 103–112, 2019.

[47] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil R. Devanur, Gregory R. Ganger, and Phillip B. Gibbons. Pipedream: Fast and efficient pipeline parallel DNN training. *CoRR*, abs/1806.03377, 2018.

[48] Taebum Kim, Hyoungjoo Kim, Gyeong-In Yu, and Byung-Gon Chun. Bpipe: Memory-balanced pipeline parallelism for training large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *ICML, Honolulu, Hawaii, USA, 23-29 July 2023*, volume 202 of *Proceedings of Machine Learning Research*, pages 16639–16653. PMLR, 2023.

[49] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: memory optimizations toward training trillion parameter models. In Christine Cuicchi, Irene Qualters, and William T. Kramer, editors, *SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, page 20. IEEE/ACM, 2020.

[50] Yanli Zhao, Andrew Gu, Rohan Varma, Liang Luo, Chien-Chin Huang, Min Xu, Less Wright, Hamid Shojanazeri, Myle Ott, Sam Shleifer, Alban Desmaison, Can Balioglu, Pritam Damania, Bernard Nguyen, Geeta Chauhan, Yuchen Hao, Ajit Mathews, and Shen Li. Pytorch FSDP: experiences on scaling fully sharded data parallel. *Proc. VLDB Endow.*, 16(12):3848–3860, 2023.

[51] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *ECCV, Amsterdam, The Netherlands, October 11-14, 2016*, volume 9908 of *Lecture Notes in Computer Science*, pages 646–661. Springer, 2016.

[52] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. In *ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[53] Minjia Zhang and Yuxiong He. Accelerating training of transformer-based language models with progressive layer dropping. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *NeurIPS, virtual, December 6-12, 2020*, 2020.

[54] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[55] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 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, 2016.

[56] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. *CoRR*, abs/1710.03740, 2017.

[57] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[58] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with warm restarts. In *ICL, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

 https://doi.org/10.52202/079017-0637

Appendices

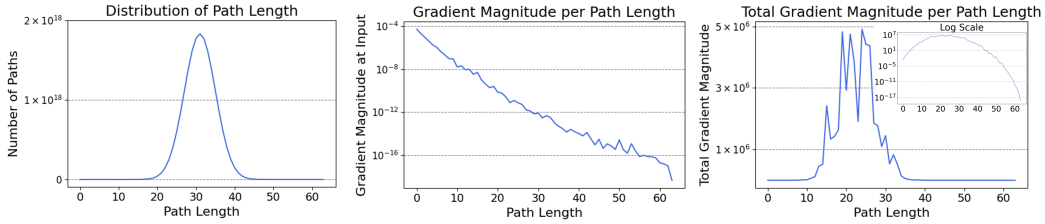
A The importance of short paths in residual networks

In Section 3.1, we interpret transformer models as a collection of numerous blocks, each composed of various modules with residual connections. Our hypothesis is that we can fine-tune LLMs well by training only certain shallow submodules. To theoretically analyze this hypothesis, we measured the impact of submodules based on their path lengths in LLaMA2-7B, as shown in Fig. 9. Specifically, we followed these steps:

- We first perform a forward pass through the entire network.
- During the backward pass, we randomly sample k residual blocks, which are back-propagated without passing through skip connections, while the remaining $n - k$ blocks are bypassed through the skip connections.
- We then measure the norm of the gradient at the input.

We take 100 measurements for each path length k. Subsequently, we multiply by the distribution of all possible path lengths, which follows a Binomial distribution, to quantify the gradient contribution from paths of a specific length.

In Fig. 9b, we observed that the gradient per path length decreases as the path length increases. Consequently, Fig. 9c demonstrates that shorter path lengths have a greater impact on the gradient in LLaMA2-7B. These observations are consistent with the existing findings [35] in ResNet [55], which attributed this phenomenon to vanishing gradients. Therefore, our DropBP enables effective training LLMs by focusing on training important short submodules.



(a) Distribution of path length. (b) Gradient per path length. (c) Total gradient per path length.

Figure 9: The impact of path length for fine-tuning LLaMA2-7B.

B Theoretical FLOPs and Actual Training Time Using DropBP

In this section, we calculate the theoretical FLOPs reduction afforded by DropBP and compare this reduction to the actual training time reduction as shown in Table 7. As outlined in Section 2, the computational costs arise from output activation calculations by Eq. 1 during forward propagation, and input and parameter gradient calculations by Eqs. 2 and 3 during backward propagation. We denote the FLOPs for these operations as F_{out}, F_{grad}, and F_{param}, respectively. Therefore, the total FLOPs for the backpropagation algorithm can be calculated by the following equation:

$$\begin{aligned} F_T &= F_{fw} + F_{bw} \\ &= F_{out} + F_{grad} + F_{param} \end{aligned} \tag{8}$$

where F_T represents the FLOPs during the entire training process, F_{fw} for forward propagation (i.e. $F_{fw} = F_{out}$), and F_{bw} for backward propagation (i.e. $F_{bw} = F_{grad} + F_{param}$). DropBP reduces FLOPs for backward propagation by a target average drop rate (p_{avg}). Therefore, total FLOPs in DropBP can be formulated as below:

$$
\begin{aligned}
F_T &= F_{fw} + (1 - p_{avg})F_{bw} \\
&= F_{out} + (1 - p_{avg})(F_{grad} + F_{param})
\end{aligned} \tag{9}
$$

Consequently, the theoretical FLOPs reduction ratio by DropBP can be represented as follow:

$$
\text{Reduction Ratio by DropBP:} \quad \frac{p_{avg}(F_{grad} + F_{param})}{F_{out} + F_{grad} + F_{param}} \tag{10}
$$

Note that in full fine-tuning (Full-FT), the computational costs for output calculations, input gradient calculations, and parameter gradient calculations are nearly identical (i.e., $F_{out} = F_{grad} = F_{param}$). Conversely, in parameter-efficient fine-tuning techniques (PEFT) such as LoRA and QLoRA, the costs of calculating parameter gradients are negligible ($F_{out} = F_{grad},\ F_{param} = 0$) due to a very small number of trainable parameters and the freezing of original LLM parameters. By substituting this into Eq. 10, the theoretical FLOPs reduction ratio by DropBP can be expressed as:

$$
\text{Reduction ratio in Full-FT:} \quad \frac{2}{3}p_{avg} \tag{11}
$$

$$
\text{Reduction Ratio in PEFT:} \quad \frac{1}{2}p_{avg} \tag{12}
$$

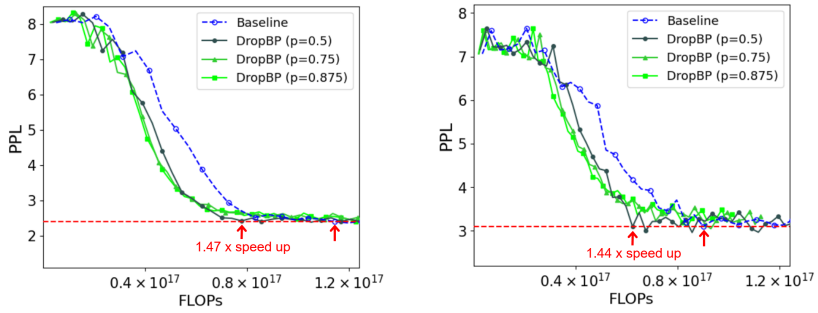
Therefore, with target average drop rates of 0.5, 0.75, and 0.875, DropBP achieves theoretical FLOPs reductions in Full-FT of 33%, 50%, and 58%, respectively, according to Eq. 11. This aligns with the actual training time reduction when utilizing DropBP in Full-FT as shown in Table 3 and Table 7. This trend is also evident when utilizing DropBP in LoRA and QLoRA. According to Eq. 12, the reductions in FLOPs for various target average drop rates of 0.5, 0.75, 0.875 are derived as 25%, 38%, and 44%, respectively. This closely aligns with the actual training time reductions observed when DropBP is applied to LoRA and QLoRA as demonstrated in Table 7.

Table 7: Training time (ms) per iteration for a sequence length of 512 through Full-FT, LoRA or QLoRA using DropBP. Mixed refers to mixed precision training [56] using BFloat16 (BF16) and 32-bit. MBS is denoted as the micro batch size. FW, BW, and Total respectively denote the time consumed for forward propagation, backward propagation, and the entire training process.

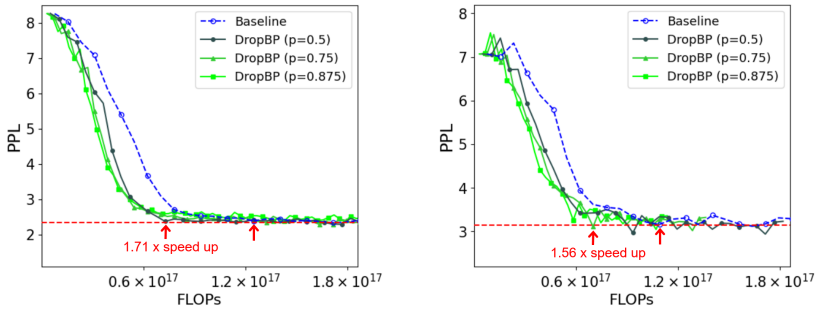
Model	Method	Precision	MBS	Drop Rate	FW	BW	Total
LLaMA2-7B	LoRA	Mixed	2	0	159	161	320
	LoRA+DropBP	Mixed	2	0.5	159	81 (-50%)	239 (-25%)
	LoRA+DropBP	Mixed	2	0.75	159	43 (-74%)	201 (-37%)
	LoRA+DropBP	Mixed	2	0.875	158	23 (-86%)	**181 (-43%)**
	Full-FT	BF16	2	0	91	192	283
	Full-FT+DropBP	BF16	2	0.5	91	98 (-49%)	189 (-33%)
	Full-FT+DropBP	BF16	2	0.75	91	52 (-73%)	143 (-50%)
	Full-FT+DropBP	BF16	2	0.875	91	30 (-85%)	**121 (-57%)**
LLaMA2-13B	LoRA	BF16	2	0	186	236	423
	LoRA+DropBP	BF16	2	0.5	186	119 (-50%)	306 (-28%)
	LoRA+DropBP	BF16	2	0.75	187	64 (-73%)	251 (-41%)
	LoRA+DropBP	BF16	2	0.875	187	33 (-86%)	**219 (-48%)**
LLaMA2-70B	QLoRA	BF16	1	0	1033	1100	2133
	QLoRA+DropBP	BF16	1	0.5	1034	566 (-49%)	1599 (-25%)
	QLoRA+DropBP	BF16	1	0.75	1033	290 (-74%)	1323 (-38%)
	QLoRA+DropBP	BF16	1	0.875	1032	158 (-86%)	**1191 (-44%)**

https://doi.org/10.52202/079017-0637

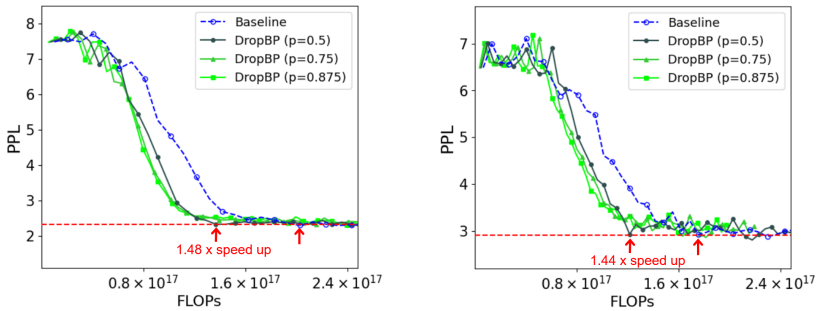
C Convergence Speed Up Using DropBP



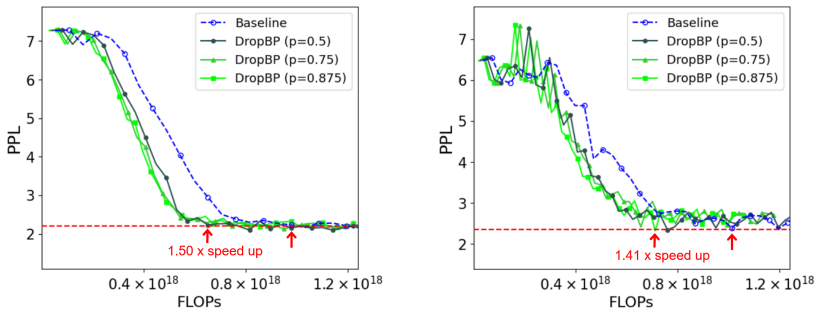
(a) LLaMA2-7B w/ LoRA (Alpaca) (b) LLaMA2-7B w/ LoRA (Dolly)



(c) LLaMA2-7B w/ Full-FT (Alpaca) (d) LLaMA2-7B w/ Full-FT (Dolly)

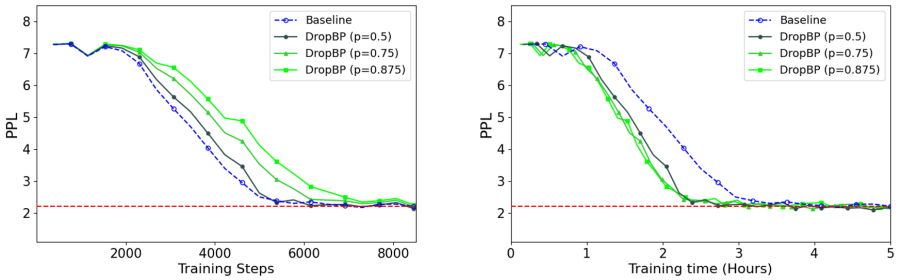


(e) LLaMA2-13B w/ LoRA (Alpaca) (f) LLaMA2-13B w/ LoRA (Dolly)



(g) LLaMA2-70B w/ QLoRA (Alpaca) (h) LLaMA2-70B w/ QLoRA (Dolly)

Figure 10: Validation perplexity (PPL) when fine-tuning LLaMA2 models through Full-FT, LoRA, or QLoRA using DropBP on the Alpaca and Dolly datasets.

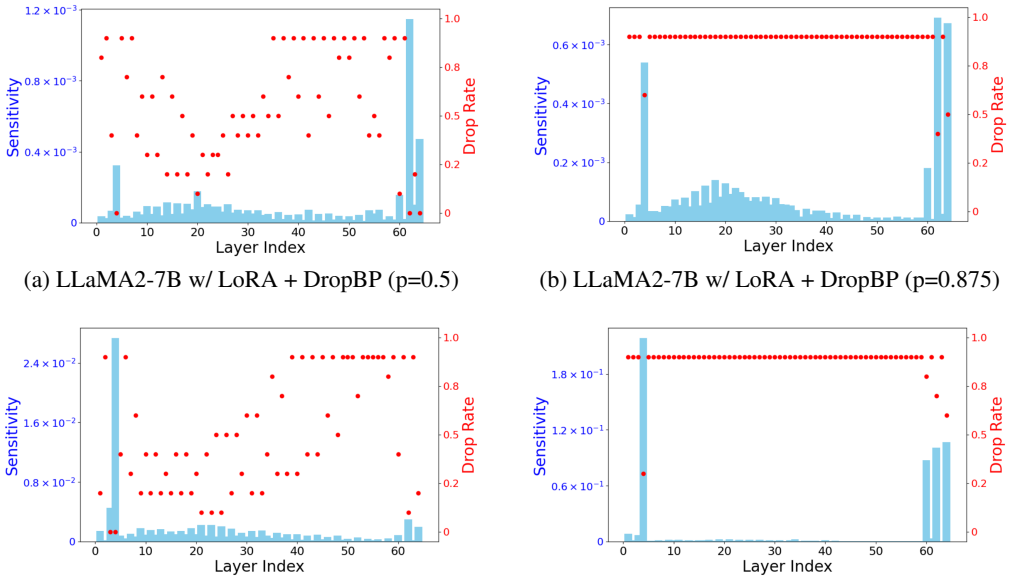


(a) Perplexity curve across training steps. (b) Perplexity curve across training time.

Figure 11: Training curves across training steps and time for fine-tuning LLaMA2-70B through QLoRA with DropBP on the Alpaca datasets.

When analyzing training curves across training steps in Fig. 11a, the convergence of loss per step at a drop rate of 0.5 is almost identical to the baseline. However, with drop rates of 0.75 and 0.875, the convergence speed per step is slower compared to baseline. Nonetheless, DropBP significantly reduces the time consumed per training step, because it skips the backward propagation computations for the dropped layers. Consequently, the convergence speed per training time is actually faster for DropBP compared to the baseline as shown in Fig. 11b.

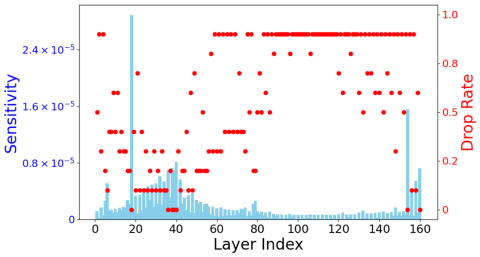
D Distribution of Drop Rates Determined by Sensitivity



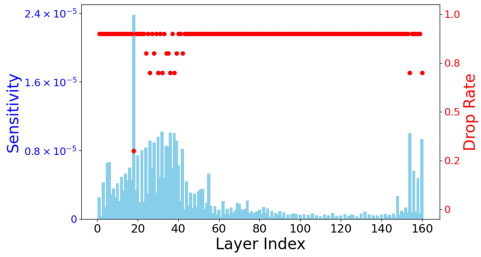
(a) LLaMA2-7B w/ LoRA + DropBP (p=0.5) (b) LLaMA2-7B w/ LoRA + DropBP (p=0.875)

(c) LLaMA2-7B w/ Full-FT + DropBP (p=0.5) (d) LLaMA2-7B w/ Full-FT + DropBP (p=0.875)

https://doi.org/10.52202/079017-0637



(e) LLaMA2-70B w/ QLoRA + DropBP (p=0.5)



(f) LLaMA2-70B w/ QLoRA + DropBP (p=0.875)

Figure 7: The distribution of drop rates determined by sensitivity when fine-tuning LLaMA2 through Full-FT, LoRA, or QLoRA using DropBP on Alpaca datasets.

E Comparisons between Layer Dropping and DropBP on fine-tuning LLMs

In this section, we compare Layerdrop (LD) [52] and Progressive Layer Dropping (PLD) [53] with DropBP under the same LLMs fine-tuning scenario. We set the relative FLOPs of LD and PLD to 0.75 of the baseline (LoRA), which corresponds to the same relative FLOPs when the drop rate of DropBP is set to 0.5.

As shown in Fig. 8, our DropBP converges faster to the same validation PPL compared to LD and PLD. Moreover, as seen in Table 8, DropBP achieves comparable accuracy to the baseline even with a relative FLOPs of 0.56, whereas LD and PLD experience a significant accuracy drop of over 5% with a relative FLOPs of 0.75. We believe this difference arises from the high sensitivity of forward propagation throughout the fine-tuning process. Specifically, layer dropping techniques omit certain layers of well-pretrained LLMs during forward propagation, resulting in significant output deviations that adversely impact the loss and the overall training process. Conversely, DropBP maintains all layers during forward propagation, thereby ensuring precise outputs and loss calculations, which facilitate stable training. Please note that, as explained in Section 5, LD and PLD are designed to accelerate the pretraining of small transformer models (SLMs) like BERT by dropping layers throughout the entire training process while DropBP only focuses on fine-tuning LLMs. In future studies, we will explore whether DropBP can similarly accelerate the pretraining of transformer models and investigate ways to improve its effectiveness.

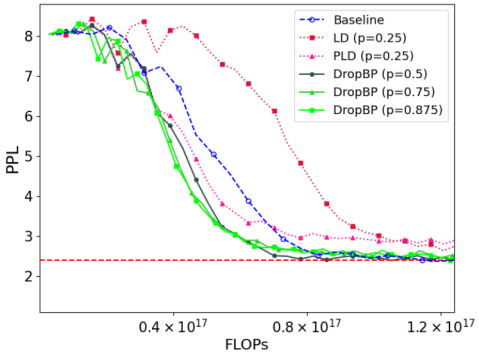


Figure 8: Validation perplexity (PPL) for fine-tuning LLaMA2-7B through LoRA (baseline) with LayerDrop (LD), Progress Layer Dropping (PLD), or DropBP on the Alpaca dataset. The p represents the target average drop rate for backward propagation in DropBP.

Table 8: Test accuracy on the 0-shot commonsense reasoning tasks when fine-tuning LLaMA2-7B through LoRA with layerdrop (LD), progressive layer dropping (PLD), and DropBP.

Method	LoRA (baseline)	LoRA+LD	LoRA+PLD	LoRA+DropBP		
				p=0.5	p=0.75	p=0.875
Relative FLOPs	1.00	0.75	0.75	0.75	0.63	**0.56**
Accuracy (%)	66.0	58.7	61.0	65.9	66.1	**66.4**

F Experimental Details

In our experimental setup, the AdamW [57] optimizer and a cosine annealing learning rate scheduler [58] were utilized as common settings. LoRA [14] and QLoRA [18] were integrated to every linear

https://doi.org/10.52202/079017-0637

layer of our model, with the LoRA parameters r and α set to 8 and 16, respectively. We experimented with all the learning rates presented in Table 9 and reported the best accuracy achieved in Table 1-2.

Table 9: Detailed Setup for Table 1-2. BS and MBS are denoted as the batch size and micro batch size, respectively. Mixed refers to mixed precision training [56] using BFloat16 (BF16) and 32-bit.

	Fine-tuning	Dataset	# Iterations	BS	MBS	Precision	Learning rate
LLaMA2-7B	LoRA	Alpaca	25K	128	2	Mixed	1e-4, 3e-4
		Dolly	7K	128	2	Mixed	1e-4, 3e-4
	Full-FT	Alpaca	25K	128	2	BF16	1e-4, 3e-4
		Dolly	7K	128	2	BF16	1e-4, 3e-4
LLaMA2-13B	LoRA	Alpaca	25K	128	2	BF16	1e-4, 3e-4
		Dolly	7K	128	2	BF16	1e-4, 3e-4
LLaMA-30B	QLoRA	Alpaca	25K	128	2	BF16	1e-4, 3e-4
		Dolly	7K	128	2	BF16	1e-4, 3e-4
LLaMA2-70B	QLoRA	Alpaca	50K	128	1	BF16	5e-5, 1e-4
		Dolly	14K	128	1	BF16	5e-5, 1e-4
LLaMA3-8B	LoRA	Oasst1	2.5K	16	4	BF16	3e-4, 5e-4

https://doi.org/10.52202/079017-0637

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We appropriately present the contributions of the paper in the Abstract and Section 1.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: In Section 5 and Appendix E, we clearly state that DropBP is developed specifically for fine-tuning, unlike other layer dropping techniques, and we indicate plans for further improvements in future research.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: In Appendix B, we mathematically calculate the theoretical reduction in FLOPs achieved by DropBP. Through experiments presented in Section 4.2 and Appendix B, we confirm that this theoretical reduction closely matches the actual training time reduction.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: In Section 4.1 and Appendix F, we provide detailed information to ensure reproducibility, and in the abstract, we present the anonymous code in Abstract to implement this.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

https://doi.org/10.52202/079017-0637

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We present the anonymous code for reproducibility in the Abstract.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide the experimental details in Section 4.1, Appendix F, and the anonymous code presented in the Abstract.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Although we conducted experiments with multiple seeds and report the average values, we did not include error bars or statistical information because the deviations were minimal, and including them would detract from the clarity of the paper.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We mention the types of devices used for the experiments in Section 4.1, and provide the time required to reproduce the experimental results in Section 4.2.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our paper adheres to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: Since we discuss a efficient fine-tuning algorithm for LLMs, we believe that our work does not have any direct negative societal impact.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.

https://doi.org/10.52202/079017-0637

- If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our work is related to the efficient fine-tuning of LLMs, and therefore, there are no specific safeguards described for the responsible release of data or models, as the nature of our research does not involve high-risk misuse scenarios.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We appropriately cite the original paper and existing codes in Section 4.1 and our code in Abstract.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.

- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We provide the new code with a proper license as an anonymized URL in the Appendix.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: Our work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Our work does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.

https://doi.org/10.52202/079017-0637

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.