Evolving Subnetwork Training for Large Language Models

Hanqi Li [1] **Lu Chen** [1 2] **Da Ma** [1] **Zijian Wu** [1] **Su Zhu** [3] **Kai Yu** [1 2]

Abstract

Large language models have ushered in a new era of artificial intelligence research. However, their substantial training costs hinder further development and widespread adoption. In this paper, inspired by the redundancy in the parameters of large language models, we propose a novel training paradigm: Evolving Subnetwork Training (EST). EST samples subnetworks from the layers of the large language model and from commonly used modules within each layer, Multi-Head Attention (MHA) and Multi-Layer Perceptron (MLP). By gradually increasing the size of the subnetworks during the training process, EST can save the cost of training. We apply EST to train GPT2 model and TinyLlama model, resulting in 26.7% FLOPs saving for GPT2 and 25.0% for TinyLlama without an increase in loss on the pre-training dataset. Moreover, EST leads to performance improvements in downstream tasks, indicating that it benefits generalization. Additionally, we provide intuitive theoretical studies based on training dynamics and Dropout theory to ensure the feasibility of EST. Our code is available at https://github.com/OpenDFM/EST.

1. Introduction

Large language models (LLMs) have become significantly larger recently, bringing tremendous potential in Natural Language Processing (NLP) tasks. The computational cost of training such large language models has become a bottleneck, hindering further development in research and applications. Additionally, the escalating hardware demands and increasing carbon footprints associated with training large language models are also crucial issues (Schwartz et al., 2020). This highlights the importance of researching efficient algorithms for training large language models.

The enormous training cost of large language models stems from their massive number of parameters. For instance, the GPT3 (Brown et al., 2020) model has 175 billion parameters, requiring 355 GPU-years and incurring a training cost of $4.6M. However, numerous studies have highlighted the redundancy in the parameters of large language models, manifested in the over-parameterization (Li et al., 2020) and conditional sparsity (Li et al., 2023b) of these models. This inspires us to optimize the training process by exploring the possibility of not training the complete model at certain stages but focusing on training subnetworks, thereby reducing the overall training cost of large language models.

In this paper, we propose a novel training paradigm, **Evolving Subnetwork Training (EST)**, towards efficient training for large language models. EST paradigm consists of two main components: 1) sample from the large language model for subnetwork training. We maintain the complete model and sample subnetworks in each training step from the model across three dimensions, including the number of attention heads, the intermediate size of multi-layer perceptron, and the total number of Transformer layers. 2) We design a sampling scheduler to plan the training process. By increasing the size of subnetworks during training and, finally, training the complete model, EST achieves training acceleration.

To demonstrate the effectiveness of EST, we first conduct experiments on GPT2 model (Radford et al., 2019), and conduct scale-up experiments on 1.1B TinyLlama (Zhang et al., 2024) model. The results show that: 1) EST saves 26.7% training cost for GPT2 model and 25.0% training cost for TinyLlama model, with a comparable loss on the pre-training dataset. 2) Models trained by EST achieve even higher downstream performance, indicating that EST benefits model generalization.

Furthermore, we dive into theoretical studies to answer the following two questions: 1) why EST can save training cost without compromising model performance? 2) Why EST can benefit model generalization? We provide a comprehensive theoretical framework based on deep learning dynamics and Dropout theory to ensure the superiority and feasibility

[1]X-LANCE Lab, Department of Computer Science and Engineering, MoE Key Lab of Artificial Intelligence, SJTU AI Institute, Shanghai Jiao Tong University, Shanghai, China [2]Suzhou Laboratory, Suzhou, China [3]AISpeech Co., Ltd., Suzhou, China. Correspondence to: Lu Chen <chenlusz@sjtu.edu.cn>, Kai Yu <kai.yu@sjtu.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

of EST.

In general, the contributions of this work include the following aspects:

- We propose a novel model training paradigm, EST, achieving higher optimization efficiency of training large language models.
- We conduct experiments on GPT2 and TinyLlama. The results show that EST saves training cost without sacrificing model performance and benefits generalization.
- We provide intuitive theoretical studies to ensure the feasibility of EST.

2. Related Work

2.1. Efficient Training for Large Language Models

Many previous works aim at improving the efficiency of training large language models, ranging from addressing low-level hardware computations and memory bottlenecks to designing high-level training strategies.

There are numerous approaches to overcome the computation bottleneck of Transformer-based models. FlashAttention (Dao et al., 2022b) identifies that the attention module is bottlenecked by memory access, and optimizes the process of attention computation, effectively reducing the training cost. Reformer (Kitaev et al., 2020) approximates attention computation based on locality-sensitive hashing and Performer (Choromanski et al., 2021) simplifies attention computation with low-rank approximation.

Sparse training methods also benefit optimization efficiency. The main component of sparse training methods is the Mixture of Experts (MoE). MoE methods (Fedus et al., 2022; Du et al., 2022) apply conditional computation according to different inputs in order to scale up models without significantly increasing training costs. The drawback of the MoE model is that its performance cannot match that of the dense model with an equivalent number of parameters. Another category of sparse training methods is based on the lottery ticket hypothesis (Frankle & Carbin, 2019; Chen et al., 2021), that certain subnetworks exhibit comparable performance to that of the original complete network. However, the sparsity generated by such methods is typically unstructured, making it challenging to translate into acceleration on general GPU devices. Monarch (Dao et al., 2022a) and Pixelated Butterfly (Dao et al., 2021) lower the training overhead of models without compromising their performance by introducing structured sparsity in matrix operations, which are more low-level and can be combined with our approach for complementary benefits. Ma et al. (2024) leverages sparsity in pre-trained LLMs to expedite the training process.

In this paper, we mainly focus on the design of a top-level training strategy, which is orthogonal to these approaches. Different from MoE methods that choose subnetworks based on input tokens, our method samples subnetworks randomly.

2.2. Incremental Training

The most similar prior works are those called incremental training methods (Shen et al., 2022). This kind of work typically starts from smaller models and gradually scales up to larger ones. Incremental training methods are effective in both the NLP and CV domains since they reduce the time needed for model training and enhance training stability. Net2net (Chen et al., 2016) first reveals the feasibility of using the parameters of smaller models as initialization for larger model parameters and provides some operations for scaling up model sizes. Shen et al. (2022) involves multi-stage training of the GPT2 model across both depth and width dimensions. bert2BERT (Chen et al., 2022) applies the principles of Net2net to pre-trained language models. MSG (Yao et al., 2023) employs a masking mechanism that sets newly expanded parameters to zero when scaling up small models. Unlike these methods that individually train smaller models, disregarding interactions between parameters, our approach EST maintains the complete model and samples subnetworks from it to train.

3. Methodology

In this section, we first review the most popular LLM architecture Transformer (Vaswani et al., 2017) in Section 3.1. In Section 3.2, we discuss how to sample subnetworks from the Transformer model for subnetwork training. In Section 3.3, we propose our efficient training paradigm, **E**volving **S**ubnetwork **T**raining (EST).

3.1. Preliminaries

Recent large language models are mainly based on Transformer architecture. Before presenting our method, we first introduce the structure of Transformer, including two main components, multi-head attention (MHA) and multi-layer perceptron (MLP).

Transformer Layer: Let $\mathbf{X}_{l-1} \in \mathbb{R}^{N \times d}$ denote the input sequence of layer l, where N is the sequence length and d denotes the hidden size. The sequence is processed iteratively by several Transformer layers with residual connection

$$\mathbf{X}_l = \mathbf{X}_{l-1} + \text{Layer}_l(\mathbf{X}_{l-1}), \forall l \in \{1, 2, ..., N_L\}, \quad (1)$$

where N_L denotes the number of Transformer layers. Each Transformer layer is composed of one MHA module and one MLP module.

MHA: MHA is used to mix information along the sequence axes to capture token-level dependencies. Let N_H denote the number of heads and d_k denote the dimension of each head. In layer l, for each head i, key, query, value projections are $\mathbf{W}^Q_{l,i}, \mathbf{W}^K_{l,i}, \mathbf{W}^V_{l,i} \in \mathbb{R}^{d \times d_k}$ and the output projection is $\mathbf{W}^O_{l,i}$. MHA is formulated as follows,

$$\mathbf{h}_{l,i} = \mathrm{softmax}\left(\frac{\mathbf{X}_{l-1}\mathbf{W}^Q_{l,i}(\mathbf{X}_{l-1}\mathbf{W}^K_{l,i})^\mathsf{T}}{\sqrt{d_k}}\right)\mathbf{X}_{l-1}\mathbf{W}^V_{l,i},$$
$$\mathbf{X}^{\mathsf{MHA}}_l = \sum_{i=1}^{N_H} \mathbf{h}_{l,i}\mathbf{W}^O_{l,i}. \tag{2}$$

MLP: MLP is used to mixes information along the hidden dimension axes. It consists of two linear layers $\mathbf{W}^1_l, \mathbf{W}^2_l \in \mathbb{R}^{d \times N_M}$ where N_M is the intermediate size of MLP. The MLP computation is

$$\mathbf{X}^{\mathsf{MLP}}_l = \sigma(\mathbf{X}^{\mathsf{MHA}}_l \mathbf{W}^1_l)(\mathbf{W}^2_l)^\mathsf{T}, \tag{3}$$

where σ is the activation function.

3.2. Subnetwork Training via Random Sampling

Subnetwork training is a training paradigm that trains a subnetwork of the model in each step rather than training the complete model. Let Φ denote the parameters of the model, $\phi \subset \Phi$ denote the parameters of the subnetwork, and L denote the loss function. Let f_ϕ denote the function of the subnetwork, which takes sequence $\mathbf{X}$ as input and outputs the prediction of next tokens. The object of subnetwork training is formulated as

$$\min_\phi L(f_\phi(\mathbf{X}), \mathbf{y}), \tag{4}$$

where $\mathbf{y}$ is the ground truth label.

In our approach, we sample subnetworks randomly from the complete model in each training step. To obtain subnetworks, we sample across three dimensions related to the size of the Transformer model: the number of attention heads N_H, the intermediate size of MLP module N_M, and the total number of Transformer layers N_L.

Sampling Attention Heads N_H: For each MHA module, during the subnetwork training, we randomly sample a subset of heads $\mathbb{I}_H$ for computation at each training step, where $\mathbb{I}_H \subset \{1, 2, ..., N_H\}$, $|\mathbb{I}_H| = N_H p_H$ and p_H is the sampling rate. Formally, during the subnetwork training, the output of MHA module is

$$\mathbf{h}_{l,i} = \mathrm{softmax}\left(\frac{\mathbf{X}_{l-1}\mathbf{W}^Q_{l,i}(\mathbf{X}_{l-1}\mathbf{W}^K_{l,i})^\mathsf{T}}{\sqrt{d_k}}\right)\mathbf{X}_{l-1}\mathbf{W}^V_{l,i},$$
$$\mathbf{X}^{\mathsf{MHA}}_l = \frac{1}{p_H}\sum_{i \in \mathbb{I}_H} \mathbf{h}_{l,i}\mathbf{W}^O_{l,i}. \tag{5}$$

The normalization operation $\frac{1}{p_H}$ is crucial as it ensures that the output distribution is consistent with the complete model. The computational cost of the MHA module in the subnetwork is reduced to a fraction p_H of that in the complete model. The detailed implementation is shown in Appendix B.1.

Sampling MLP Intermediate Size N_M: For each MLP module, we sample the intermediate dimension, *i.e.*, the columns of $\mathbf{W}^1_l$ and $\mathbf{W}^2_l$, at each training step. Let $\mathbb{I}_M \subset \{1, 2, ..., N_M\}$ denote the index of sampled columns where $|\mathbb{I}_M| = N_M p_M$ and p_M is the sampling rate. We select columns in $\mathbb{I}_M$ from the $\mathbf{W}^1_l$ and $\mathbf{W}^2_l$ to obtain $\hat{\mathbf{W}}^1_l$ and $\hat{\mathbf{W}}^2_l$. The subnetwork's MLP computation is

$$\mathbf{X}^{\mathsf{MLP}}_l = \frac{1}{p_M}\sigma(\mathbf{X}^{\mathsf{MHA}}_l \hat{\mathbf{W}}^1_l)(\hat{\mathbf{W}}^2_l)^\mathsf{T}. \tag{6}$$

Similarly, the the output of MLP module requires normalization to ensure the consistency of the output distribution. The computational cost of the MLP module during subnetwork training is reduced to a fraction p_M. The detailed implementation is shown in Appendix B.2.

Sampling Transformer The Number of Layer N_L: We employ a sampling strategy similar to Stochastic Depth (Huang et al., 2016), randomly skipping some layers of the Transformer model. Let p_L denote the sampling rate. We sample from Transformer layers to obtain a subset $\mathbb{I}_L \subset \{1, 2, ..., N_L\}$ where $|\mathbb{I}_L| = N_L p_L$. For each layer in the Transformer, we compute the layer output as

$$\mathbf{X}_l = \begin{cases} \mathbf{X}_{l-1} + \mathrm{Layer}_l(\mathbf{X}_{l-1}), & \text{if } l \in \mathbb{I}_L \\ \mathbf{X}_{l-1}, & \text{if } l \notin \mathbb{I}_L \end{cases}. \tag{7}$$

During subnetwork training, the computational cost of the subnetwork can be reduced to a fraction p_L of the complete model, for only $N_L p_L$ layers are activated.

3.3. Evolving Subnetwork Training

In this section, we propose our novel training paradigm for large language models, **E**volving **S**ubnetwork **T**raining (EST), which applies subnetwork training method in Section 3.2, and progressively increases the size of subnetworks. Finally, train the complete model.

Definition 3.1. Let T denote the total stages of training. A sampling scheduler consists of two parts: 1) $S = (s^1, ..., s^T)$ that contains the time points of stage transitions, indicating when to increase the size of subnetworks; 2) $P = [(p^1_H, p^1_M, p^1_L), ..., (p^T_H, p^T_M, p^T_L)]$ that contains sampling rates in each stage, indicating how to increase the size of subnetworks.

EST employs the sampling scheduler to plan the training process. By incrementally increasing the size of subnet-

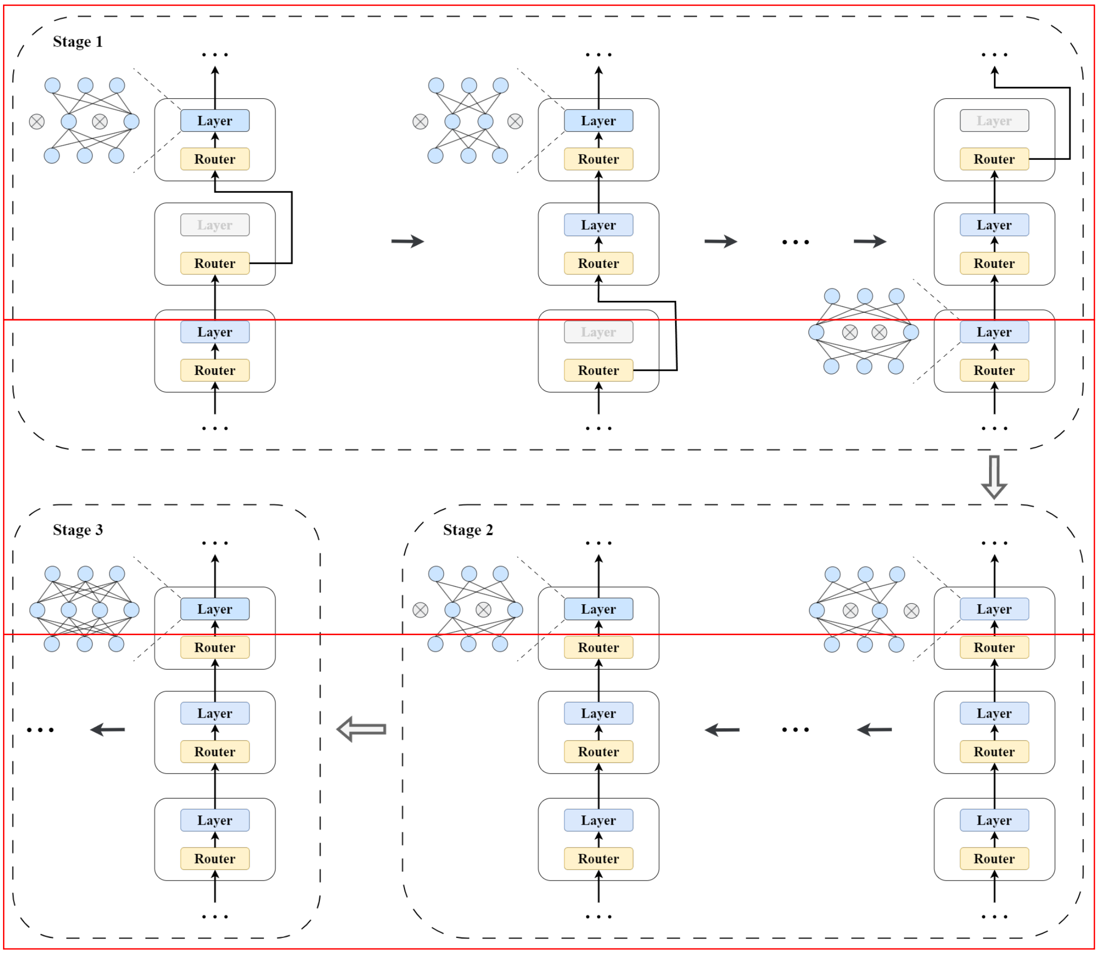

Figure 1. Overview of our EST method with practical sampling scheduler. The router takes $\mathbb{I}_L$ as input and determines whether to activate the current layer. In stage 1, we obtain a subnetwork to train by sampling from N_H, N_M and N_A dimensions. In such subnetworks, only some layers are activated and in each activated layer, and only some attention heads and MLP neurons are used. In stage 2, all layers are activated while in each layer still only a subset of the layer is used. In stage 3, the complete model is activated.

works as the training stages progress and eventually training the complete model, EST achieves training acceleration. The pseudo-code of EST is as Algorithm 1.

Practical Sampling Scheduler: In this paper, we won't dive into complex sampling schedulers. For convenience, we use a kind of three-stage sampling scheduler in practice. Specifically, our sampling scheduler consists of the following three stages:

- **Stage 1**: In this stage, we sample from all three dimensions to achieve the highest acceleration ratio. That is, $0 < p_H < 1, 0 < p_M < 1, 0 < p_L < 1$.
- **Stage 2**: In this stage, we stop sampling from the Transformer layers, ensuring that the number of activated layers in the subnetwork is consistent with the complete model, while continuing sampling from the MHA and MLP modules. That is, $0 < p_H < 1, 0 < p_M < 1$, $p_L = 1$. Additionally, in this stage, we keep p_H and p_M consistent with the first stage.
- **Stage 3**: In the final stage, train the complete model, where $p_H = 1, p_M = 1, p_L = 1$.

The process of EST with this kind of sampling scheduler is illustrated in Figure 1.

Training Cost Saving: Assuming that, under the condition of equal training steps, the model trained by EST achieves the same performance as the original model, EST can indeed achieve a reduction in training cost. This is because the cost of training subnetworks is smaller than that of training the complete model.

For ease of analysis, we calculate how much training cost is saved by EST with the practical sampling scheduler. Let C_H, C_M denote the cost of each MHA module and MLP

Algorithm 1 Evolving Subnetwork Training

Input: Dataset $(\mathcal{X}, \mathcal{Y})$, sampling scheduler $S = (s^1, ..., s^T)$, $P = [(p_H^1, p_M^1, p_L^1), ..., (p_H^T, p_M^T, p_L^T)]$.

Randomly initialize the model.
for $t = 1 \to T$ **do** {Training stages}
  for $k = s^{t-1} \to s^t$ **do** {Training steps, $s^0 = 0$}
    Sample $(\mathbf{X}, \mathbf{y}) \sim (\mathcal{X}, \mathcal{Y})$.
    Sample $I_L \subset \{1, 2, ..., N_L\}, |I_L| = p_L N_L$.
    $\mathbf{X}_0 = \text{EMBEDDING}(\mathbf{X})$.
    for $l = 1 \to N_L$ **do** {Transformer layers}
      if $l \notin \mathbb{I}_L$ **then**
        $\mathbf{X}_l = \mathbf{X}_{l-1}$.
        Continue.
      end if
      Sample $I_H \subset \{1, 2, ..., N_H\}, |I_H| = p_H N_H$.
      Sample $I_M \subset \{1, 2, ..., N_M\}, |I_M| = p_M N_M$.
      Compute $\mathbf{X}_l^{\text{MHA}}$ condition on $\mathbb{I}_M$.
      Compute $\mathbf{X}_l^{\text{MLP}}$ condition on $\mathbb{I}_H$.
      $\mathbf{X}_l = \mathbf{X}_{l-1} + \mathbf{X}_l^{\text{MLP}}$.
    end for
    $\hat{\mathbf{y}} = \text{PROJECTION}(\mathbf{X}_{N_L})$.
    Compute loss with $L(\hat{\mathbf{y}}, \mathbf{y})$.
    Backward and optimize.
  end for
end for

module, neglecting other modules like Layer Normalization (Ba et al., 2016) as their cost is relatively small compared to MHA and MLP. So the training cost of a single training step in each stage is formulated as

$$\begin{aligned} C_1 &= N_L p_L (p_H C_H + p_M C_M), \\ C_2 &= N_L (p_H C_H + p_M C_M), \\ C_3 &= N_L (C_H + C_M). \end{aligned} \tag{8}$$

Based on the training steps for each stage, the total training cost can be calculated. Let r_1, r_2, r_3 denote training steps of each stage, respectively. The total training cost is formulated as

$$\begin{aligned} C_{EST} &= r_1 C_1 + r_2 C_2 + r_3 C_3 \\ &= (r_1 N_L p_L p_H + r_2 N_L p_H + r_3 N_L) C_H \\ &+ (r_1 N_L p_L p_M + r_2 N_L p_M + r_3 N_L) C_M. \end{aligned} \tag{9}$$

On the other hand, the training cost of training the complete model is

$$C_{original} = (r_1 + r_2 + r_3) N_L (C_H + C_M). \tag{10}$$

For a more intuitive illustration, Table 1 shows the training cost under specific configurations that $p_H = p_M = p_L = 0.5$, $r_1 = r_2 = r_3 = r$ compared with cost of training the original model through naive training method. Under such configurations, EST can save 41.7% of training cost.

Table 1. An intuitive example of training cost saving. The real world wall time saving is shown in Appendix A.3

Stages	EST	Original
Stage 1	$0.25 r N_L (C_H + C_M)$	$r N_L (C_H + C_M)$
Stage 2	$0.5 r N_L (C_H + C_M)$	$r N_L (C_H + C_M)$
Stage 3	$r N_L (C_H + C_M)$	$r N_L (C_H + C_M)$
Total	$1.75 r N_L (C_H + C_M)$	$3 r N_L (C_H + C_M)$
Saving	$1.25 r N_L (C_H + C_M)$	0

4. Experiments

In this section, we first present our main results with GPT2 model on the in-domain pre-train task and out-domain downstream tasks in Section 4.1. In addition, to show the scalability of our approach, we conduct experiments with TinyLlama model in Section 4.2.

4.1. Main Results with GPT2

Experiment Setup: We conduct experiments with GPT2-base model, which has 117M parameters in total, pre-trained on OpenWebText dataset (Radford et al., 2019) with AdamW optimizer (Loshchilov & Hutter, 2019) from scratch. The batch size is set to 512 and the sequence length is 1024. The total training step is 150k. For the downstream performance, we experiment on three tasks: GLUE (Wang et al., 2018), SQuAD (Rajpurkar et al., 2016) , and LAMBADA (Paperno et al., 2016).

For GPT2-base model, the practical sampling scheduler is set to $S = (20k, 70k, 150k)$ and $P = [(0.5, 0.5, 0.5), (0.5, 0.5, 1), (1, 1, 1)]$, which saves 26.7% computation cost of training.

We choose Staged Training (Shen et al., 2022), which is a kind of incremental training method and has two stages, as a baseline. In stage 1, Staged Training method trains the model with half hidden size. At the end of stage 1, expand the parameters of the model. In stage 2, train the complete model. The stage transition occurs at step 50k, and this baseline saves 16.7% of the training FLOPs.

For another baseline MSG (Yao et al., 2023), due to the inability of this method to simultaneously expand attention heads and intermediate sizes, the training stage settings in the MSG baseline differ slightly from our EST method:

- **Stage1 (0-20k)**: Utilizing a model with half the number of layers, attention heads, and intermediate size.
- **Stage2 (20k-40k)**: Restoring the number of layers and using a model with half the attention heads and intermediate size.

- **Stage3 (40k-70k)**: Restoring the attention heads and using a model with half the intermediate size.
- **Stage4 (70k-150k)**: Training the complete model.

Main Results: We compare EST with the naive training method that trains the complete model, Staged Training method and MSG method. The results are as the Table 2. Our approach EST saves 26.7% of training FLOPs and leads to 1.22x speed up of the wall clock training time without increasing the loss on the validation dataset. The loss curve of GPT2 trained by EST can be found in Appendix A.1. Additionally, we find that the model trained by EST has better downstream performance, indicating that EST also enhances the generalization of GPT2 model. However, despite saving 16.7% of the training FLOPs, the performance of the model obtained by Staged Training cannot match the original model. Compared to MSG method, our EST approach achieves higher acceleration effects and delivering superior model performance.

Effect of Sampling Scheduler: We also conduct experiments to evaluate the effect of different sampling schedulers. Besides our practical sampling scheduler, we evaluate model performance on five different sampling schedulers:

- EST-ONE-STAGE: $S = (150k)$, $P = [(0.5, 0.5, 1)]$.
- EST-TWO-STAGE-A: $S = (50k, 150k)$, $P = [(0.5, 0.5, 1), (1, 1, 1)]$.
- EST-TWO-STAGE-B: $S = (70k, 150k)$, $P = [(0.5, 0.5, 1), (1, 1, 1)]$.
- EST-TWO-STAGE-C: $S = (90k, 150k)$, $P = [(0.5, 0.5, 1), (1, 1, 1)]$.
- EST-THREE-STAGE: $S = (20k, 70k, 150k)$, $P = [(0.5, 0.5, 0.5), (1, 1, 0.5), (1, 1, 1)]$.

Among these five sampling schedulers, EST-ONE-STAGE is exactly the stage 2 of the practical sampling scheduler and does not train the complete model at all. EST-TWO-STAGE-A, EST-TWO-STAGE-B, and EST-TWO-STAGE-C skip the stage 1 of the practical sampling scheduler and have different stage transition points. EST-THREE-STAGE, on the other hand, modifies the stage 2 of the practical sampling scheduler by disabling the sampling from MHA and MLP and enabling the sampling from the number of layers. The results are shown in Table 3.

We find that our three-stage practical sampling scheduler saves more training cost than two-stage sampling schedulers with comparable model performance. On the other hand, results of EST-ONE-STAGE and EST-TWO-STAGE-C indicate that a long enough stage 3 is vital. In addition, the experiment on EST-THREE-STAGE sampling scheduler indicates that, in stage 2, sampling from MHA modules and MLP modules performs better than sampling from layers.

4.2. Scale-up to TinyLlama

Experiment Setup: We pre-train a 1.1B TinyLlama model with 22 layers on the subset of SlimPajama dataset (Soboleva et al., 2023) and Starcoder dataset (Li et al., 2023a) from scratch, using AdamW optimizer. The batch size is set to 1024 and the sequence length is 2048. The total training step is 60k, containing 130B tokens in total. We report the validation loss on SlimPajama dataset and downstream performance on GPT4All (Anand et al., 2023) benchmarks. GPT4All contains seven different datasets, evaluating the few-shot common sense reasoning ability of models.

For TinyLlama model, the practical sampling scheduler is set to $S = (10k, 25k, 60k)$ and $P = [(0.5, 0.5, 0.5), (0.5, 0.5, 1), (1, 1, 1)]$, which saves 25.0% computation cost of training.

Main Results: We demonstrate the scalability of EST on TinyLlama in Table 4. Compared with the original model trained by the naive training method, EST method saves 25.0% of training FLOPs and leads to 1.22x speed up of wall clock training time, with comparable loss. The loss curve of TinyLlama trained by EST can be found in Appendix A.2. In addition, model generalization performance, measured by the average score of GPT4All, is improved by EST.

5. Theoretical Studies

In this section, we aim to answer two key questions: 1) Why can EST method save training cost without compromising model performance? 2) Why do models trained using the EST method exhibit better generalization performance? We study the training dynamics of EST in Section 5.1 to answer question 1 and study the loss landscape of models trained using the EST method in Section 5.2 to answer question 2.

5.1. Why Can EST Save Training Cost without Compromising Model Performance

In general, EST enhances the dynamics of model training, resulting in a steeper loss curve and faster loss descent compared to the naive training approach, so it can save training cost without compromising model performance. To provide a more specific explanation, we first need to introduce two important properties in previous incremental training methods (Shen et al., 2022): *loss-preserving property* and *training-dynamics-preserving property*. Subsequently, we point out that it is precisely because EST breaks away from these two properties that it can achieve savings in model training cost in a broad range of scenarios.

Table 2. Main results of the experiment with GPT2 model. We choose Staged Training (Shen et al., 2022) and MSG (Yao et al., 2023) baselines. Loss is evaluated on the validation dataset. For metrics, we use accuracy and F1 score for SQuAD, and accuracy for LAMBADA. The detailed results of GLUE are in Appendix A.1.

	WALL CLOCK TIME(HOURS)	SPEED UP	SAVING FLOPS
ORIGINAL	185.0	1X	0
STAGED TRAINING	173.1	1.06X	16.7%
MSG	160.8	1.16X	24.4%
EST	**151.6**	**1.22X**	**26.7%**

	AVERAGE GLUE	SQUAD	LAMBADA	LOSS
ORIGINAL	79.84	66.74/77.06	29.44	3.06
STAGED TRAINING	73.82	61.65/72.71	28.99	3.15
MSG	79.88	62.05/71.89	29.06	3.13
EST	**80.66**	**67.14/77.15**	**32.01**	**3.05**

Table 3. Ablation study on different sampling schedulers. We find that our proposed practical sampling scheduler strikes a good balance between model performance and FLOPs saving. Compared to other types of schedulers, our practical sampling scheduler performs the best with the same training cost.

	SAVING FLOPS	LOSS	AVERAGE GLUE	SQUAD	LAMBADA
ORIGINAL	0	3.06	79.84	66.74/77.06	29.44
EST-ONE-STAGE	**50.0%**	3.36	77.78	63.95/74.65	26.57
EST-TWO-STAGE-A	16.7%	**3.04**	**81.05**	**67.39/77.76**	29.59
EST-TWO-STAGE-B	23.3%	3.06	80.17	67.18/77.51	29.71
EST-TWO-STAGE-C	30.0%	3.09	80.41	66.55/77.01	28.68
EST-THREE-STAGE	26.7%	3.07	79.47	65.73/76.22	29.67
EST	26.7%	3.05	80.66	67.14/77.15	**32.01**

Loss-preserving Property: The loss-preserving property implies that during a transition in the training stages, the models before and after the transition should represent the same function, resulting in identical loss.

Training-dynamics-preserving Property: Intuitively, the training-dynamics-preserving property means that in the final stage of incremental training, the loss curve should match that of the target model.

Why Should Break Away from These Properties: In incremental training methods, maintaining these two properties is to ensure the feasibility of extending the parameters of a smaller model as the initialization parameters for the target model. However, to maintain these two properties while achieving the goal of saving training cost, incremental training methods require expanding the parameters to the size of the target model early in the training process (Shen et al., 2022). Therefore, when the model training requires a sufficient number of training steps, incremental training methods can actually save very little in training cost. EST breaks away from these two properties, alleviating this issue.

Break Away from Loss-preserving Property: EST method dose not maintain the loss-preserving property. During stage transitions when increasing the size of the subnetworks, the loss experiences a sudden drop compared to the previous stage.

Due to the equivalence between the random sampling of subnetworks and Structural Dropout (Pal et al., 2020), we can theoretically demonstrate using Dropout theory. Intuitively, training subnetworks via random sampling implicitly introduces a regularization term to the loss function, and the increase in subnetwork size reduces this regularization term, resulting in a sudden drop in loss.

Break Away From Training-dynamics-preserving Property: EST method does not maintain the training-dynamics-preserving property. In the final stage of training, the model trained with EST exhibits better training dynamics compared to the target model obtained through naive training method.

Intuitively, EST method is equivalent to the use of Structural Dropout in earlier stages. Early Dropout pushes model

Table 4. Main results of experiment with TinyLlama model. Loss is evaluated on the validation dataset. The detailed results of GPT4All are shown in Appendix A.2.

	WALL CLOCK TIME(HOURS)	SPEED UP	SAVING FLOPS	LOSS	GPT4ALL
ORIGINAL	192.8	1x	0	**2.64**	42.40
EST	**158.2**	**1.22x**	**25%**	2.65	**42.79**

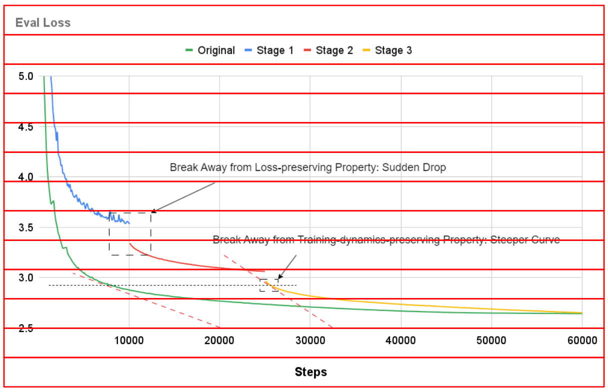

Figure 2. Loss curves of EST compared with the original training method.

parameters into a flatter region of the loss landscape. Even in the final stage when training the complete model, the model parameters can still maintain flatness, reducing the difficulty of parameter optimization and accelerating the descent of the loss.

Towards Training Cost Saving: How does the disruption of these two properties affect the efficiency of model training? We compare the loss curves of TinyLlama trained by EST and the original training method in Figure 2.

Firstly, we observe that during stage transitions, the loss drops sharply, which is the result of breaking the loss-preserving property, providing a better starting point for each stage. On the other hand, we notice that in stage 3, the slope of the EST loss curve is greater than the slope of the original loss curve under the same loss condition. This is a direct result of breaking the training-dynamics-preserving property, which is the key to the success of EST. If the training-dynamics-preserving property holds, the loss curve in stage 3 will be parallel to the loss curve of the original training method, and thus, the two curves will not intersect, eliminating the possibility of saving training costs. It is precisely because the EST method improves the dynamics of model training that it leads to savings in training costs without sacrificing model performance.

5.2. Why Can EST Benefit Model Generalization

In Section 4, we observe that the EST method not only achieves comparable loss on the pre-training dataset compared to the naive training method, but also brings some

Table 5. The trace of Hessian matrix and the average GLUE score of model trained through naive training method and through EST method. With almost the same pre-training loss, EST method has higher GLUE score for smaller trace of Hessian matrix.

	Loss	GLUE	$\mathrm{Tr}[\nabla^2 L(\phi)]$
Original	3.06	73.89	11763
EST	3.05	74.66	2510

improvement in downstream tasks. This indicates an enhancement in the model's generalization performance.

Liu et al. (2023a) investigate the phenomenon where different models exhibit significant differences in downstream tasks under the same loss on the pre-training dataset. Furthermore, they find a strong correlation between the model's generalization ability and the trace of the Hessian matrix of the loss function with respect to the model parameters.

The process of sampling subnetworks in the EST method is equivalent to Structural Dropout, which can effectively reduce the trace of the Hessian matrix in the early stages. However, more importantly, we find that even in the final stage of training the complete model, the Hessian matrix still maintains a relatively small trace until the end of training. This contributes to the better generalization performance of the final model obtained through EST.

In Table 5, we demonstrate that GPT2 models trained through EST exhibit a smaller trace of the Hessian matrix and stronger generalization performance compared to models obtained through naive training method. Here $\mathrm{Tr}[\nabla^2 L(\phi)]$ denotes the trace of Hessian matrix of the loss function with respect to the model parameters.

6. Conclusion

Our goal is to achieve more efficient training for large language models. We propose a novel training method, Evolving Subnetwork Training (EST), which operates subnetwork training via random sampling and uses sampling scheduler to plan the process of training incrementally. Our approach enhances the efficiency of model training on GPT2 and TinyLlama models, saving 26.7% and 25.0% of training FLOPs respectively, with comparable pre-training performance. In addition, EST benefits the generalization ability of both GPT2 and TinyLlama, evaluated by several downstream tasks. We also provide intuitive theory studies, demonstrating the feasibility and superiority of EST. Through theoretical analysis, we find that the efficient training dynamics of EST comes from the flatness of parameters. This insight may inspire other efficient training methods. In future works, we aim to provide the theoretical support for the design of sampling schedulers, to apply EST for training on even larger models. Additionally, since EST essentially samples for matrix multiplication, it can be applied not only to Transformer models but also to models like Mamba (Gu & Dao, 2023). We will conduct experiments on other types of models to broaden the application scope of EST.

Impact Statement

This work's ethical impact is rooted in the ethical risks associated with large language models themselves. While there are numerous ethical risks linked to large language models, this paper primarily focuses on efficient training for such models, and thus, these ethical risks are not the main emphasis of this paper. Therefore, we believe it is not necessary to highlight them here.

The future societal consequences of this work primarily involve its impact on the environment and the applications of large language models. As this work helps reduce the training costs of large language models, it contributes to mitigating the carbon emissions caused by research and applications of such models, thereby aiding environmental conservation. Simultaneously, the cost savings may also facilitate a broader and more widespread application of large language models in society.

Acknowledgments

This work is funded by the China NSFC Projects (92370206, U23B2057, 62106142 and 62120106006) and Shanghai Municipal Science and Technology Major Project (2021SHZDZX0102).

References

Anand, Y., Nussbaum, Z., Duderstadt, B., Schmidt, B., and Mulyar, A. Gpt4all: Training an assistant-style chatbot with large scale data distillation from gpt-3.5-turbo. https://github.com/nomic-ai/gpt4all, 2023.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chen, C., Yin, Y., Shang, L., Jiang, X., Qin, Y., Wang, F., Wang, Z., Chen, X., Liu, Z., and Liu, Q. bert2bert: Towards reusable pretrained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2134–2148, 2022.

Chen, T., Goodfellow, I. J., and Shlens, J. Net2net: Accelerating learning via knowledge transfer. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.05641.

Chen, X., Cheng, Y., Wang, S., Gan, Z., Wang, Z., and Liu, J. Early{bert}: Efficient {bert} training via early-bird lottery tickets, 2021. URL https://openreview.net/forum?id=I-VfjSBzi36.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

Dao, T., Chen, B., Liang, K., Yang, J., Song, Z., Rudra, A., and Re, C. Pixelated butterfly: Simple and efficient sparse training for neural network models. *arXiv preprint arXiv:2112.00029*, 2021.

Dao, T., Chen, B., Sohoni, N. S., Desai, A., Poli, M., Grogan, J., Liu, A., Rao, A., Rudra, A., and Ré, C. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pp. 4690–4721. PMLR, 2022a.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022b.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pp. 5547–5569. PMLR, 2022.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJl-b3RcF7`.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces, 2023.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Deep networks with stochastic depth. volume 9908, pp. 646–661, 10 2016. ISBN 978-3-319-46492-3. doi: 10.1007/978-3-319-46493-0_39.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=rkgNKkHtvB`.

Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Davaadorj, M., Lamy-Poirier, J., Monteiro, J., Shliazhko, O., Gontier, N., Meade, N., Zebaze, A., Yee, M.-H., Umapathi, L. K., Zhu, J., Lipkin, B., Oblokulov, M., Wang, Z., Murthy, R., Stillerman, J., Patel, S. S., Abulkhanov, D., Zocca, M., Dey, M., Zhang, Z., Fahmy, N., Bhattacharyya, U., Yu, W., Singh, S., Luccioni, S., Villegas, P., Kunakov, M., Zhdanov, F., Romero, M., Lee, T., Timor, N., Ding, J., Schlesinger, C., Schoelkopf, H., Ebert, J., Dao, T., Mishra, M., Gu, A., Robinson, J., Anderson, C. J., Dolan-Gavitt, B., Contractor, D., Reddy, S., Fried, D., Bahdanau, D., Jernite, Y., Ferrandis, C. M., Hughes, S., Wolf, T., Guha, A., von Werra, L., and de Vries, H. Starcoder: may the source be with you! 2023a.

Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., and Gonzalez, J. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *International Conference on machine learning*, pp. 5958–5968. PMLR, 2020.

Li, Z., You, C., Bhojanapalli, S., Li, D., Rawat, A. S., Reddi, S., Ye, K., Chern, F., Yu, F., Guo, R., et al. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *Conference on Parsimony and Learning (Recent Spotlight Track)*, 2023b.

Liu, H., Xie, S. M., Li, Z., and Ma, T. Same pre-training loss, better downstream: Implicit bias matters for language models. In *International Conference on Machine Learning*, pp. 22188–22214. PMLR, 2023a.

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Re, C., et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023b.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=Bkg6RiCqY7`.

Ma, D., Chen, L., Wang, P., Xu, H., Li, H., Sun, L., Zhu, S., Fan, S., and Yu, K. Sparsity-accelerated training for large language models. In *The 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL `https://openreview.net/forum?id=HvofKj7jlC`.

Pal, A., Lane, C., Vidal, R., and Haeffele, B. D. On the regularization properties of structured dropout. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7671–7679, 2020.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, N. Q., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. In Erk, K. and Smith, N. A. (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL `https://aclanthology.org/P16-1144`.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. SQuAD: 100,000+ questions for machine comprehension of text. In Su, J., Duh, K., and Carreras, X. (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL `https://aclanthology.org/D16-1264`.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

Shen, S., Walsh, P., Keutzer, K., Dodge, J., Peters, M., and Beltagy, I. Staged training for transformer language models. In *International Conference on Machine Learning*, pp. 19893–19908. PMLR, 2022.

Soboleva, D., Al-Khateeb, F., Myers, R., Steeves, J. R., Hestness, J., and Dey, N. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, 2023. URL `https://huggingface.co/datasets/cerebras/SlimPajama-627B`.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Yao, Y., Zhang, Z., Li, J., and Wang, Y. 2x faster language model pre-training via masked structural growth. *arXiv preprint arXiv:2305.02869*, 2023.

Zhang, P., Zeng, G., Wang, T., and Lu, W. Tinyllama: An open-source small language model, 2024.

References

Anand, Y., Nussbaum, Z., Duderstadt, B., Schmidt, B., and Mulyar, A. Gpt4all: Training an assistant-style chatbot with large scale data distillation from gpt-3.5-turbo. https://github.com/nomic-ai/gpt4all, 2023.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33: 1877–1901, 2020.

Chen, C., Yin, Y., Shang, L., Jiang, X., Qin, Y., Wang, F., Wang, Z., Chen, X., Liu, Z., and Liu, Q. bert2bert: Towards reusable pretrained language models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2134–2148, 2022.

Chen, T., Goodfellow, I. J., and Shlens, J. Net2net: Accelerating learning via knowledge transfer. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.05641.

Chen, X., Cheng, Y., Wang, S., Gan, Z., Wang, Z., and Liu, J. Early{bert}: Efficient {bert} training via early-bird lottery tickets, 2021. URL https://openreview.net/forum?id=I-VfjSBzi36.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking attention with performers. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ua6zuk0WRH.

Dao, T., Chen, B., Liang, K., Yang, J., Song, Z., Rudra, A., and Re, C. Pixelated butterfly: Simple and efficient sparse training for neural network models. *arXiv preprint arXiv:2112.00029*, 2021.

Dao, T., Chen, B., Sohoni, N. S., Desai, A., Poli, M., Grogan, J., Liu, A., Rao, A., Rudra, A., and Ré, C. Monarch: Expressive structured matrices for efficient and accurate training. In *International Conference on Machine Learning*, pp. 4690–4721. PMLR, 2022a.

Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems*, 35:16344–16359, 2022b.

Du, N., Huang, Y., Dai, A. M., Tong, S., Lepikhin, D., Xu, Y., Krikun, M., Zhou, Y., Yu, A. W., Firat, O., et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International Conference on Machine Learning*, pp. 5547–5569. PMLR, 2022.

Fedus, W., Zoph, B., and Shazeer, N. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *The Journal of Machine Learning Research*, 23(1):5232–5270, 2022.

Frankle, J. and Carbin, M. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=rJl-b3RcF7.

Gu, A. and Dao, T. Mamba: Linear-time sequence modeling with selective state spaces, 2023.

Huang, G., Sun, Y., Liu, Z., Sedra, D., and Weinberger, K. Deep networks with stochastic depth. volume 9908, pp. 646–661, 10 2016. ISBN 978-3-319-46492-3. doi: 10.1007/978-3-319-46493-0_39.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

Li, R., Allal, L. B., Zi, Y., Muennighoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., Liu, Q., Zheltonozhskii, E., Zhuo, T. Y., Wang, T., Dehaene, O., Davaadorj, M., Lamy-Poirier, J., Monteiro, J., Shliazhko, O., Gontier, N., Meade, N., Zebaze, A., Yee, M.-H., Umapathi, L. K., Zhu, J., Lipkin, B., Oblokulov, M., Wang, Z., Murthy, R., Stillerman, J., Patel, S. S., Abulkhanov, D., Zocca, M., Dey, M., Zhang, Z., Fahmy, N., Bhattacharyya, U., Yu, W., Singh, S., Luccioni, S., Villegas, P., Kunakov, M., Zhdanov, F., Romero, M., Lee, T., Timor, N., Ding, J., Schlesinger, C., Schoelkopf, H., Ebert, J., Dao, T., Mishra, M., Gu, A., Robinson, J., Anderson, C. J., Dolan-Gavitt, B., Contractor, D., Reddy, S., Fried, D., Bahdanau, D., Jernite, Y., Ferrandis, C. M., Hughes, S., Wolf, T., Guha, A., von Werra, L., and de Vries, H. Starcoder: may the source be with you! 2023a.

Li, Z., Wallace, E., Shen, S., Lin, K., Keutzer, K., Klein, D., and Gonzalez, J. Train big, then compress: Rethinking model size for efficient training and inference of transformers. In *International Conference on machine learning*, pp. 5958–5968. PMLR, 2020.

Li, Z., You, C., Bhojanapalli, S., Li, D., Rawat, A. S., Reddi, S., Ye, K., Chern, F., Yu, F., Guo, R., et al. The lazy neuron phenomenon: On emergence of activation sparsity in transformers. In *Conference on Parsimony and Learning (Recent Spotlight Track)*, 2023b.

Liu, H., Xie, S. M., Li, Z., and Ma, T. Same pre-training loss, better downstream: Implicit bias matters for language models. In *International Conference on Machine Learning*, pp. 22188–22214. PMLR, 2023a.

Liu, Z., Wang, J., Dao, T., Zhou, T., Yuan, B., Song, Z., Shrivastava, A., Zhang, C., Tian, Y., Re, C., et al. Deja vu: Contextual sparsity for efficient llms at inference time. In *International Conference on Machine Learning*, pp. 22137–22176. PMLR, 2023b.

Loshchilov, I. and Hutter, F. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Ma, D., Chen, L., Wang, P., Xu, H., Li, H., Sun, L., Zhu, S., Fan, S., and Yu, K. Sparsity-accelerated training for large language models. In *The 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://openreview.net/forum?id=HvofKj7jlC.

Pal, A., Lane, C., Vidal, R., and Haeffele, B. D. On the regularization properties of structured dropout. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 7671–7679, 2020.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, N. Q., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. In Erk, K. and Smith, N. A. (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1525–1534, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1144. URL https://aclanthology.org/P16-1144.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. SQuAD: 100,000+ questions for machine comprehension of text. In Su, J., Duh, K., and Carreras, X. (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 2383–2392, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1264. URL https://aclanthology.org/D16-1264.

Schwartz, R., Dodge, J., Smith, N. A., and Etzioni, O. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

Shen, S., Walsh, P., Keutzer, K., Dodge, J., Peters, M., and Beltagy, I. Staged training for transformer language models. In *International Conference on Machine Learning*, pp. 19893–19908. PMLR, 2022.

Soboleva, D., Al-Khateeb, F., Myers, R., Steeves, J. R., Hestness, J., and Dey, N. SlimPajama: A 627B token cleaned and deduplicated version of RedPajama, 2023. URL https://huggingface.co/datasets/cerebras/SlimPajama-627B.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

Yao, Y., Zhang, Z., Li, J., and Wang, Y. 2x faster language model pre-training via masked structural growth. *arXiv preprint arXiv:2305.02869*, 2023.

Zhang, P., Zeng, G., Wang, T., and Lu, W. Tinyllama: An open-source small language model, 2024.

A. Additional Experiment Details

A.1. Details for GPT2 Experiment

Details for Pre-training: We pre-train 117M GPT2 model on OpenWebText dataset from scratch, using AdamW optimizer. Batch size is set to 512 and each example contains 1024 tokens. The initial learning rate is set to 6×10^{-4}, followed by a linear learning rate decay. The pre-training loss curves of EST method on training dataset and validation dataset are as Figure 3.

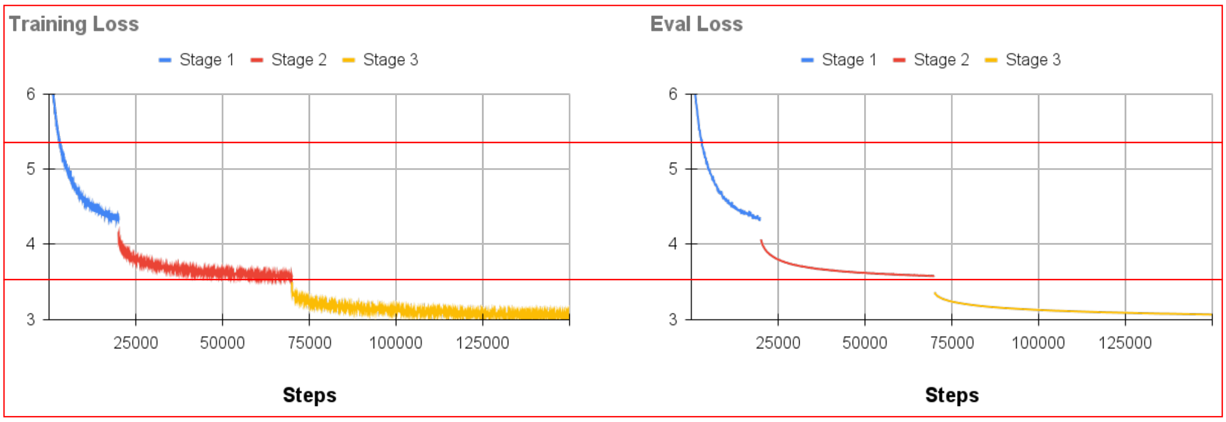

Figure 3. Training and evaluation loss of EST training with GPT2-base model.

Details for GLUE benchmark: The detailed scores evaluated on GLUE are as Table 6. CoLA is measured by Matthews correlation and accuracy. STS-B is measured by Pearson/Spearman correlation. MRCP and QQP are measured by accuracy and F1 score. Others are measured by accuracy.

Table 6. Detailed GLUE scores of GPT2-base model.

	Saving FLOPs	CoLA	SST-2	MRPC	STS-B
Original	0	43.21/77.27	90.02	80.15/86.39	86.63/86.25
Staged Training	16.7%	14.50/65.68	87.15	76.25/84.43	83.97/83.76
EST-one-stage	50.0%	37.27/74.88	89.91	76.23/83.81	82.62/82.36
EST-two-stage-A	16.7%	44.61/78.04	91.51	83.58/88.39	87.11/86.97
EST-two-stage-B	23.3%	43.26/77.09	91.06	80.39/86.44	86.56/86.57
EST-two-stage-C	30.0%	45.70/78.14	89.79	80.15/86.43	86.71/86.52
EST-three-stage	26.7%	42.66/76.22	91.28	80.16/86.48	84.33/84.12
EST	26.7%	45.88/78.04	91.51	80.88/87.13	86.69/86.55

	Saving FLOPs	QQP	MNLI(m/mm)	QNLI	RTE
Original	0	89.50/85.83	79.14/79.59	86.07	67.87
Staged Training	16.7%	86.91/82.36	75.45/75.79	82.44	61.01
EST-one-stage	50.0%	89.05/85.26	77.81/78.29	86.16	67.51
EST-two-stage-A	16.7%	89.39/85.70	80.24/80.47	86.94	70.76
EST-two-stage-B	23.3%	89.55/85.80	80.02/80.73	86.49	68.23
EST-two-stage-C	30.0%	89.40/85.82	80.36/80.08	86.88	69.31
EST-three-stage	26.7%	89.25/85.49	79.01/80.13	86.25	67.79
EST	26.7%	89.27/85.53	80.38/80.81	86.99	68.95

A.2. Details for TinyLlama Experiment

Details for Pre-training: We pre-train TinyLlama 1.1B model on the subset of SlimPajama dataset and Starcoder dataset, consisting of 130B tokens. We use AdamW optimizer, and the max learning rate is set to 4×10^{-4} with 2000 warm-up steps, followed by a cosine learning rate decay. The pre-training loss curves of EST method on training dataset and validation dataset are as Figure 4.

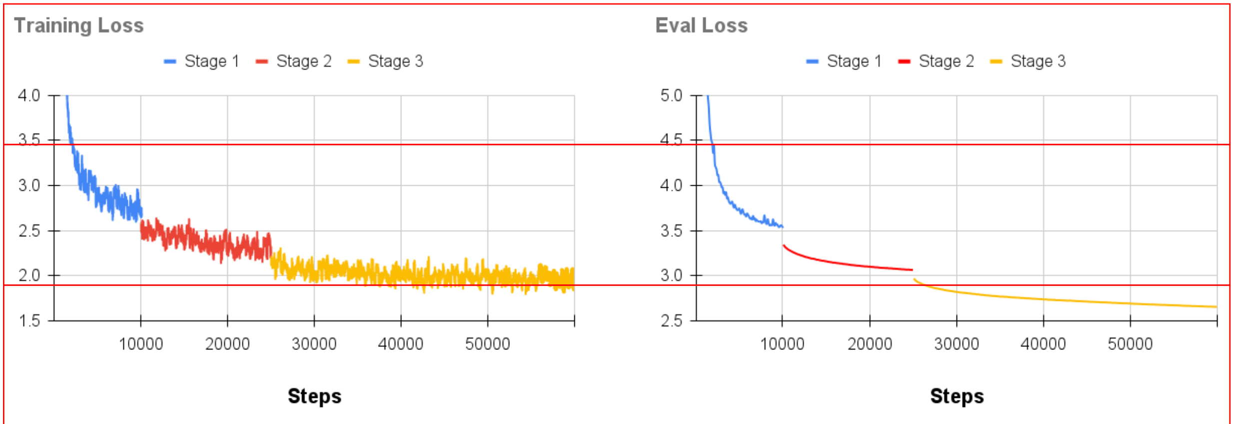

Figure 4. Training and evaluation loss of EST training with GPT2-base model.

Details for GPT4All benchmark: The detailed scores evaluated on GPT4All are as Table 7.

Table 7. Detailed GPT4All scores of TinyLlama model.

	SAVING FLOPS	HELLASWAG	OBQA	WINOGRANDE	ARC_c	ARC_e	BOOLQ	PIQA
ORIGINAL	0	33.54	29.40	50.51	23.04	38.55	59.60	62.13
EST	25.0%	33.40	27.20	52.88	23.29	38.93	61.16	62.68

A.3. Details for Wall Time Saving

We test the efficiency of the EST method on GPT2 and TinyLlama models and assess the real acceleration effects. We will analyze the wall time overhead of each module and the time overhead under different training setups. Here we mainly analyze the impact of sampling on the MHA and MLP modules. These two modules involve matrix multiplication, and our sampling alters the size of these matrices. Since matrix multiplication is parallelized on GPUs, it's challenging to intuitively calculate the actual acceleration effect. For both GPT2-base and TinyLlama 1.1B model, we investigate the impact of different batch sizes on training speed when using Distributed Data Parallel (DDP). For simplicity, we discuss the practical sampling scheduler in Table 1. We use A100 80GB GPU to test both GPT2 model and TinyLlama model.

For the GPT2 model, the actual acceleration effects are as Table 8. For the TinyLlama 1.1B model, the actual acceleration effects are as Table 9. In these two tables, GPU time refers to the time spent on forward computations for each module or layer on the GPU. Total time indicates the overall time cost for each training step, including both forward and backward computation.

The final speedup rate is not $4\times$ for stage1 and not $2\times$ for stage 2 due to two reasons: (1) GPUs compute matrix multiplication in parallel, so the time consumption is not directly proportional to the number of rows or columns of the matrix; (2) In addition to GPU computation time, there is also high memory access overhead during model training. As the batch size increases, the bottleneck of training gradually shifts from memory access to computation, resulting in an increase in the speedup, and the speedup on GPU time gradually approaches $2\times$.

Table 8. Wall time overhead of GPT2 model. Each example contains 1024 tokens.

Micro Batch Size	8	16	32	48
GPU time (ms) of MHA	2.51	4.84	9.48	14.12
GPU time (ms) of EST MHA	1.41	2.57	4.92	7.27
GPU time (ms) of MLP	0.92	1.78	3.38	5.01
GPU time (ms) of EST MLP	0.59	1.05	1.95	2.86
GPU time (ms) of Transformer Layer	3.56	6.86	13.31	19.77
GPU time (ms) of EST Transformer Layer	2.12	3.86	7.34	10.79
Total time (ms) of Stage 1 training step	182.49	278.62	447.91	525.55
Total time (ms) of Stage 2 training step	210.59	309.27	497.37	721.37
Total time (ms) of Stage 3 (original) training step	211.05	388.46	646.85	1065.23

Table 9. Wall time overhead of TinyLlama 1.1B model. Each example contains 2048 tokens.

Micro Batch Size	1	2	4	8
GPU time (ms) of MHA	1.43	2.10	3.93	8.14
GPU time (ms) of EST MHA	1.45	1.63	2.24	4.11
GPU time (ms) of MLP	0.39	0.69	1.43	2.87
GPU time (ms) of EST MLP	0.44	0.61	0.98	1.60
GPU time (ms) of Transformer Layer	2.05	3.12	5.99	11.88
GPU time (ms) of EST Transformer Layer	2.50	2.57	3.85	6.84
Total time (ms) of Stage 1 training step	288.37	228.66	360.24	501.61
Total time (ms) of Stage 2 training step	374.05	296.95	399.80	659.46
Total time (ms) of Stage 3 (original) training step	274.23	336.71	555.82	915.40

B. Implementation Details

Among the three sampling methods we use, sampling for the number of Transformer layers is straightforward and will not be elaborated. However, the sampling operation for the dimensions of MHA and MLP modules within each layer is more complex. This will be detailed here. The operation within each Transformer layer can be illustrated as Figure 5. The index generator generates indexes $\mathbb{I}_H$, $\mathbb{I}_M$ and $\mathbb{I}_L$. The router before each module takes $\mathbb{I}_H$ or $\mathbb{I}_M$ as input and activates the corresponding part of the module.

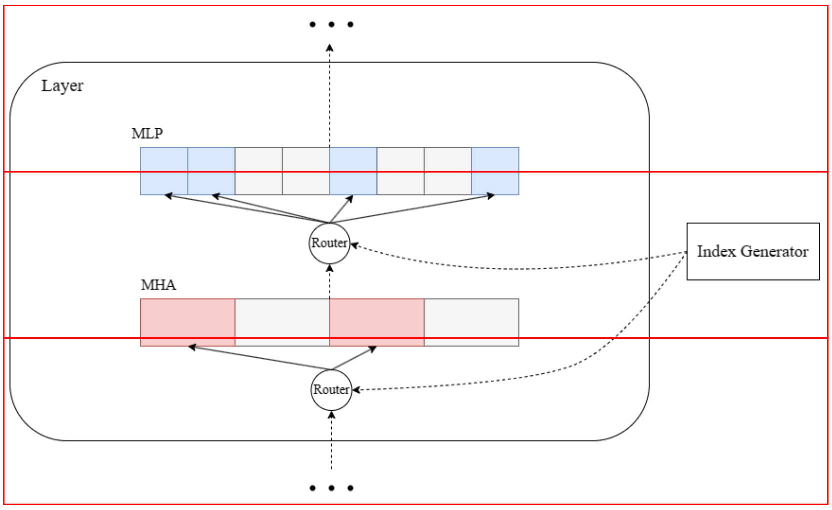

Figure 5. Computation in each Transformer layer during subnetwork training.

B.1. Implementation of Sampling for MHA module

For the MHA module, we sample a subset of heads for computation. Specifically, this involves sampling along the dimensions of the output projection matrices $\mathbf{W}^Q$,$\mathbf{W}^K$,$\mathbf{W}^V$, and selecting the corresponding input dimensions in the output matrix $\mathbf{W}^O$. The detailed process is illustrated in Figure 6.

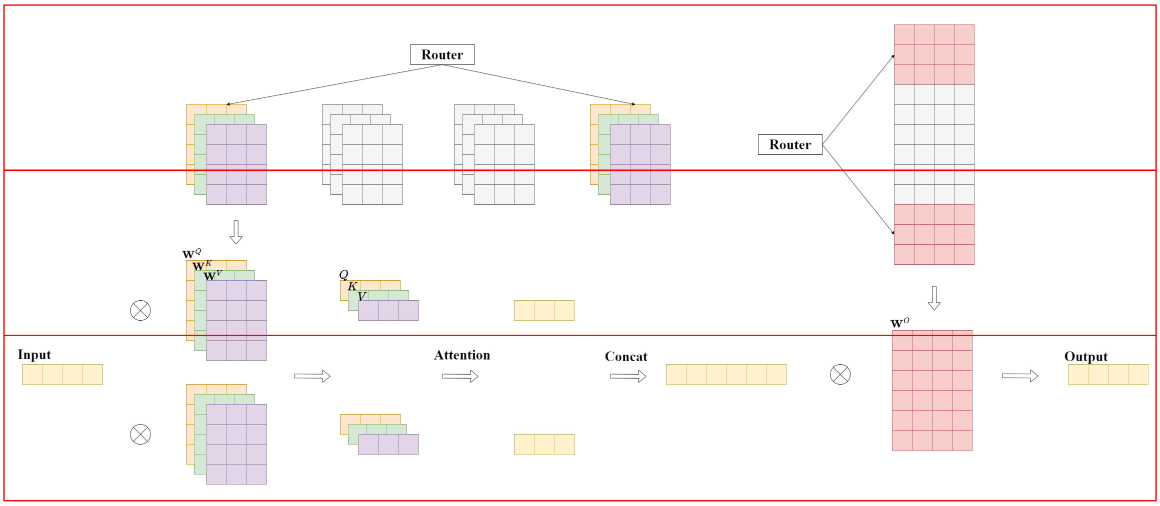

Figure 6. The detailed implementation of sampling for MHA module.

B.2. Implementation of Sampling for MLP module

For the MLP module, we sample columns from $\mathbf{W}^1$ and $\mathbf{W}^2$ for computation. The detailed process is illustrated in Figure 7.

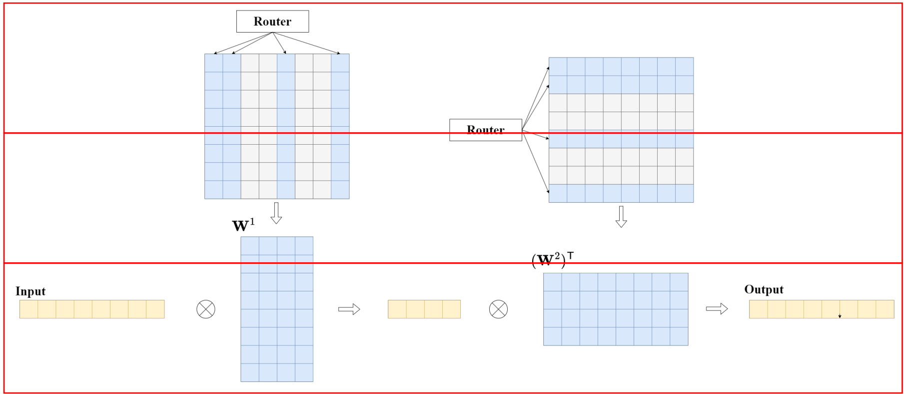

Figure 7. The detailed implementation of sampling for MLP module.

Unlike in Deja Vu (Liu et al., 2023b), in our training scenario, since our sampling operation is performed per batch rather than per token, the cost of extracting rows and columns from the matrices is relatively small, and kernel fusion is not necessary.

B.3. Implementation of Index Generator

The index generator simply generates random numbers as sampling indices. However, since it operates on the CPU, and once the indices are generated, they need to be transferred to the GPU memory. Executing it as part of the model before each forward pass could result in unnecessary time overhead. To optimize the training process as much as possible, we use an additional thread to run the index generator asynchronously to the model training. Once the index generator generates the next set of indices, it places them in a queue. When the model needs to sample, it retrieves the values from the queue. This completely eliminates the overhead of the index generator.