EFTNAS: Searching for Efficient Language Models in First-Order Weight-Reordered Super-Networks

J. Pablo Muñoz[1], **Yi Zheng**[2], **Nilesh Jain**[1]
[1]Intel Labs, Santa Clara, CA, USA
[2]Intel Corporation, Beijing, CN
{pablo.munoz, yi.zheng, nilesh.jain}@intel.com

Abstract

Transformer-based models have demonstrated outstanding performance in natural language processing (NLP) tasks and many other domains, e.g., computer vision. Depending on the size of these models, which have grown exponentially in the past few years, machine learning practitioners might be restricted from deploying them in resource-constrained environments. This paper discusses the compression of transformer-based models for multiple resource budgets. Integrating neural architecture search (NAS) and network pruning techniques, we effectively generate and train weight-sharing super-networks that contain efficient, high-performing, and compressed transformer-based models. A common challenge in NAS is the design of the search space, for which we propose a method to automatically obtain the boundaries of the search space and then derive the rest of the intermediate possible architectures using a first-order weight importance technique. The proposed end-to-end NAS solution, EFTNAS, discovers efficient subnetworks that have been compressed and fine-tuned for downstream NLP tasks. We demonstrate EFTNAS on the General Language Understanding Evaluation (GLUE) benchmark and the Stanford Question Answering Dataset (SQuAD), obtaining high-performing smaller models with a reduction of more than 5x in size without or with little degradation in performance.

Keywords: Language Models, Neural Architecture Search, Transformers

1. Introduction

Transformers have driven the recent advancements in Artificial Intelligence. For instance, they are at the core of many successful *large language models (LLMs)*, recently capturing the public's attention. However, it is more than large models that have been successful. Small and medium-sized transformer-based models power many everyday artificial intelligence applications. Unfortunately, these models cannot often be deployed in compute-constrained environments, e.g., many edge devices, because of their size, computing, or memory requirements. The *attention* operator (see Section 2 for details) at the core of Transformers has $O(n^2)$ computational and memory complexity in the input sequence length, which has motivated research on how to effectively compress these models using traditional techniques like pruning, quantization and neural architecture search (NAS). Another of the many research paths explores approximations of the *attention* operator (Tay et al., 2022) or alternative architectures, e.g., using *Long Convolutions* and strengthening the data-control path of the proposed architectures (Poli et al., 2023).

A standard workflow in Transformer-based architectures is to have a model trained with a large dataset and then fine-tune this model for a downstream task, e.g., question-answering on a smaller dataset (Raffel et al., 2020). This paper focuses on techniques for obtaining efficient and compressed Transformer-based models using weight-sharing NAS super-networks that are fine-tuned for a downstream task. We demonstrate that super-network-based NAS is a practical approach to obtaining smaller, more efficient transformers-based models. However, NAS solutions are plagued with many challenges. For instance, designing a good search space is a challenging task. Another challenge is related to the effective reordering of weights and the strategy for sampling subnetworks during training. This paper tackles these challenges and discusses the following contributions: A novel approach, EFTNAS, for (1) **automating the generation of the NAS search space** using unstructured weight importance information, effectively bridging the gap between unstructured pruning and weight-sharing super-network elasticity, and (2) **improving the weight arrangement of the super-network**, resulting in robust super-networks with high-performing subnetworks that we compare to other approaches to confirm the benefits of the proposed approach.

This paper is organized as follows: Section 2 provides a background and related work references for Transformers, Neural Network Pruning, Knowledge Distillation, and Neural Architecture Search. Section 3 focuses on the proposed methods for obtaining high-performing Transformer-based subnetworks. Section 4 discusses results obtained with EFTNAS. Section 5 presents some concluding remarks, and Sections 6 and 7 discuss limitations and ethical considerations.

LREC-COLING 2024, pages 5596–5608
20-25 May, 2024.

2. Related Work

Transformers Since their inception, Transformers (Vaswani et al., 2017) and the *attention* operator have become the preferred components of Deep Learning models and workflows. Transformer-based models have excelled at natural language processing (NLP) tasks (Devlin et al., 2019; Liu et al., 2020; Goyal et al., 2021) and in many other domains, e.g., image classification (Liu et al., 2021), image segmentation (Zhang et al., 2023), multi-modal schemes (Xu et al., 2023). At these models' core is a stack of Transformers blocks, each with two main components: the *attention* mechanism and a fully connected feed-forward network. The transformer's paper by Vaswani et al. adopts *scaled dot-product attention* (Equation 1).

$$\text{Attention}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V}, \quad (1)$$

where $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$ are the result of linearly transforming the input, $\boldsymbol{X}$, i.e., text embeddings or the output of the previous Transformer block, depending on the location of the Transformer block in the stack, with the weight matrices $\boldsymbol{W}^Q$, $\boldsymbol{W}^K$, $\boldsymbol{W}^V$, i.e., $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}^Q$, $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}^K$, $\boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}^V$. d_k is the hidden dimensionality for $\boldsymbol{Q}$ and $\boldsymbol{K}$. The scaling factor, $\sqrt{d_k}$, ensures the Softmax operation does not saturate. Multiple attention "heads" are expected to run in parallel, denoted as *multi-head attention* (MHA).

$$\begin{aligned}\text{MHA}(\boldsymbol{Q},\boldsymbol{K},\boldsymbol{V}) &= \text{Concat}(\text{head}_1, ..., \text{head}_h)\boldsymbol{W}^O \\ \text{head}_i &= \text{Attention}(\boldsymbol{Q}\boldsymbol{W}_i^Q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V)\end{aligned} \quad (2)$$

Following the *attention* layers, each transformer block has a feed-forward network (FFN), which is usually composed of linear projection layers with an activation function, often the *Gaussian Error Linear Unit (GELU)* (Hendrycks and Gimpel, 2016). These components are complemented with residual connections and layer normalization operations. The reader can find more details about the transformer architecture in Vaswani et al. (2017). Section 3 describes the steps taken by EFTNAS to design a search space based on a pre-trained transformer-based model and add *elasticity* (defined later) to selected transformer blocks, resulting in the generation of weight-sharing super-networks with smaller and more efficient models, a.k.a. subnetworks.

Neural Network Pruning is a popular method for compressing neural networks and reducing their computational complexity. The goal is to remove parameters without significantly affecting the model's final performance (LeCun et al., 1989). *Unstructured* pruning works at the parameter level without any constraints, but it is often difficult to see its benefits due to the lack of support in generally available hardware. On the other hand, *structured pruning* can be better realized in many hardware platforms. To determine which elements to remove or mask (*pruning criteria*), a common zeroth-order approach is to use *magnitude pruning* and calculate the $lp-$norm ($\|\mathbf{x}\|_p := (\Sigma_{i=1}^n |x_i|^p)^{\frac{1}{p}}$, where $p \geq 1$), and remove the elements with a value below a threshold. In the case of Transformer-based models, first-order pruning methods, e.g., *movement pruning* (Sanh et al., 2020) and its extension, *block pruning* (Lagunas et al., 2021), have been shown to outperform traditional magnitude pruning algorithms. We refer the reader to Blalock et al. (2020) for a comprehensive survey on pruning algorithms.

Knowledge Distillation A popular technique to improve the performance of compressed models is to use a larger model, the $teacher$, to influence the training of a smaller model, the $student$ (Hinton et al., 2015). We compare our approach against several popular approaches that use *knowledge distillation* to obtain high-performing compressed models. DistilBERT (Sanh et al., 2019) trains a distilled version of BERT (Devlin et al., 2019) while removing components and reducing the number of layers. DistilBERT retains most of BERT's performance while also performing well on downstream tasks. TinyBERT (Jiao et al., 2020) proposes a two-stage distillation framework (general and task-specific) in which the second stage is improved with data augmentation. TinyBERT improves compared to the results obtained by DistilBERT. MiniLM (Wang et al., 2020b) focuses on distilling the self-attention component of the teacher's last Transformer block. This approach is further improved by introducing *multi-head self-attention relations* in MiniLMv2 (Wang et al., 2021). EFTNAS' subnetworks compete with the compressed models obtained from these approaches (Section 4).

Weight-Sharing Super-Networks and Neural Architecture Search (NAS) Given a set of possible neural network architectures (*search space*), NAS methods apply *search* and *performance estimation* strategies to discover high-performing architectures that are often smaller and more efficient than human-crafted architectures (Elsken et al., 2019). Many NAS techniques have been proposed to discover high-performing architectures (White et al., 2023). One-shot weight-sharing approaches, e.g., (Bender et al., 2018; Cai et al., 2019; Guo et al., 2020; Liu et al., 2018b,a; Pham et al., 2018; Yu and Huang, 2019; Cai et al., 2020) have shown to be effective, avoiding the pitfalls of early NAS approaches, e.g., training many candidates, either partially or entirely, from scratch. Several techniques have been proposed to train the generated super-networks, e.g., *Progressive Shrinking* (Cai et al., 2020) and *Single Stage* (Yu et al., 2020) training. The automatic generation of weight-sharing super-networks has been demonstrated by BootstrapNAS (Muñoz et al., 2022).

NAS has been used in the past to compress Transformer-based models, e.g., HAT (Wang et al., 2020a), demonstrated how to generate a Transformer-based super-network for machine translation. NAS-BERT (Xu et al., 2021) produced a super-network with efficient subnetworks that were demonstrated using the GLUE benchmark (Wang et al., 2019). AutoDistil (Xu et al., 2022) proposes to mitigate the problem of subnetwork interference during training by partitioning the search space into many subspaces and training a super-network for each subspace using knowledge distillation. However, researchers are still confronting the challenges of designing robust search spaces and improving the robustness of the trained super-networks. Next, we present EFTNAS, an approach to automate the generation of the search space using first-order weight importance information that can be adjusted based on the target's resource budget. To further boost the robustness of EFTNAS' super-networks, the weight importance information is reused to reorder the super-network weights and further improve the quality of the super-network.

3. Methodology

Figure 1 illustrates EFTNAS' stages to obtain compressed high-performing transformer-based models, a.k.a. subnetworks, for a particular task. In the following sections, We describe (1) the generation of the unstructured weight importance mask (Section 3.1), (2) a method for the automated generation of the search space (Section 3.2), (3) training of the super-network, and the subsequent search for high-performing subnetworks that achieve a performance target for a particular task (Section 3.3).

3.1. From Unstructured Weight Importance to Structured Super-Network Elasticity

Previous NAS approaches have used zeroth-order weight importance, e.g., $lp-$norm, to sort the weights of the super-network to allow smaller subnetworks to benefit from more robust shared weights (Cai et al., 2020). Empirically (Section 4.3), we observe and demonstrate the benefits of discarding the zeroth-order approach in favor of first-order weight importance (Sanh et al., 2020) for weight-sharing super-networks. There are several steps to obtain information on weight importance. For each layer of interest, an importance mask, $\mathbf{M}$, often a binary mask, is computed using a threshold τ and a score $\mathbf{S}$, i.e., $\mathbf{M} = 1(\mathbf{S} > \tau)$ (Sanh et al., 2020; Lagunas et al., 2021). $\mathbf{S}$ is computed over several t forward and backward passes (Equation 3). $\boldsymbol{W}$ are the weights of the layer of interest, $\mathcal{L}$ is the loss function, and α_S is a scaling factor for the movement accumulator, $\mathbf{S}$.

$$\mathbf{S}_{i,j}^{(T)} = -\alpha_S \sum_{t<T} \left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{i,j}} \right)^{(t)} \boldsymbol{W}_{i,j}^{(t)} \qquad (3)$$

EFTNAS uses first-order weight importance information in two novel ways. First, EFTNAS uses both the binary mask, $\mathbf{M}$, and the score, $\mathbf{S}$, to automate the generation of the search space (Section 3.2), and second, $\mathbf{S}$ is analyzed to reorder the weights received from the pre-trained model before optimizing the generated super-network (Section 3.3). Next, we discuss EFTNAS' approach to generating the NAS search space automatically.

3.2. Automated Design of the Search Space based on the Desired Subnetwork Computational Complexity

A common challenge when using neural architecture search is the design of the search space, i.e., the set of possible architectures (subnetworks) that can be *activated*, used to update the weights of the super-network, and then extracted as compressed models. Additional challenges include determining the values for other NAS hyper-parameters, e.g., the minimum possible width of a layer and the intervals between possible configurations, to name a few. When we can activate different configurations in a layer, we say the layer is *elastic* (Muñoz et al., 2022).

A naive approach to designing the search space is to collect all the possible configurations of every layer with a variable configuration, e.g., different numbers of heads in the multi-head attention layer. Using this example, a challenge with this naive approach is determining the minimum (and maximum) number of attention heads that should be allowed or the possible width of the subsequent intermediate layers in the feed-forward network. Unfortunately, these naive approaches often result in large search spaces that are impractical for NAS due to their immense exploration costs. The problems are compounded when the NAS solution enters the search stage for high-performing subnetworks in subpar search spaces, spending search cycles on search space regions that might contribute poorly to the overall objective.

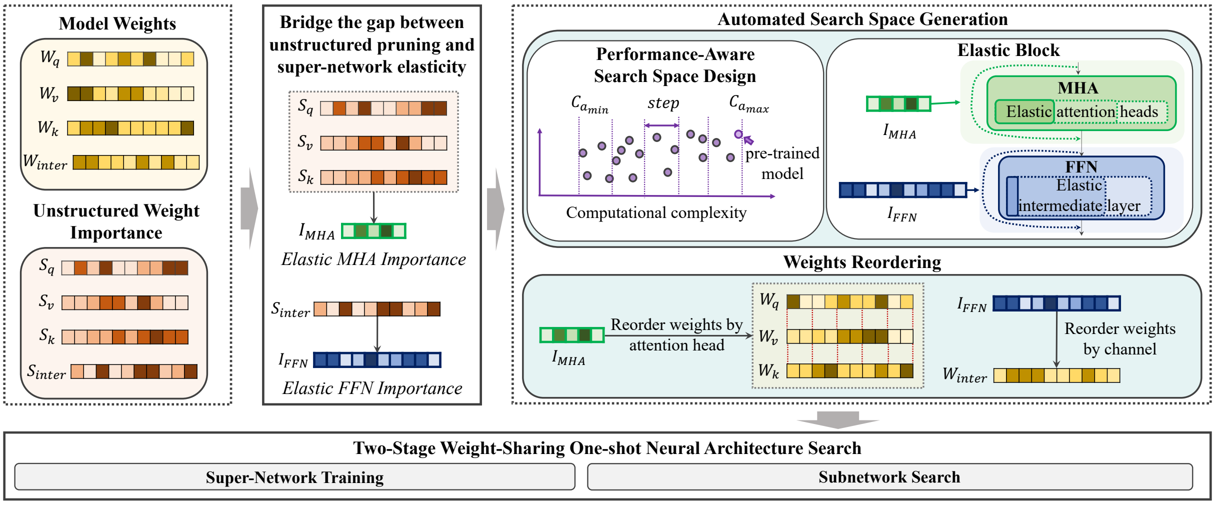

Figure 1: EFTNAS' end-to-end workflow. Unstructured weight importance information is used to obtain subnetwork configurations at the boundaries of the desired search space. Next, intermediate subnetworks are identified depending on the required complexity for the search space, and all subnetwork configurations are combined, effectively automating the search space generation based on the desired computation complexity. Weights are reordered, elasticity is enabled, and the super-network is trained. Finally, high-performing Transformer-based subnetworks are discovered for a variety of performance targets.

Performance-Aware Search Space Design As detailed in Algorithm 1, EFTNAS first obtains an importance score, $\mathbf{S}_m$ (Equation 3), for the weights of each layer in a pre-trained model, m. EFTNAS uses the weight importance information stored in the binary mask, $\mathbf{M}_m$ (obtained from $\mathbf{S}_m$ using the value of a threshold, τ), and the desired computational complexities, $C(a_{min})$ and $C(a_{max})$ (more details below) **to obtain the corresponding subnetwork configurations for the boundaries and intermediate points of the search space**.

At each iteration, EFTNAS searches for the value of τ that will result in a binary mask, $\mathbf{M}_m$, for the whole model corresponding to a subnetwork with computational complexity, c, e.g., a particular measurement in GFLOPs. Other metrics can be used, e.g., the latency of the subnetwork in a particular target hardware device. Binary search is an effective method to find this value quickly, and EFTNAS allows for approximations to speed up the search for the value of τ. The corresponding subnetwork configuration is stored in a set $\mathcal{B}$. The explored range for c starts at $c = C(a_{min})$, the computational complexity of the minimal subnetwork, a_{min}, i.e., the architectural configuration that satisfies the lower end of the desired computational complexity range. EFTNAS continues to find the corresponding N-2 intermediate subnetwork configurations that satisfy the steps in the required computational complexity until the architectural configuration of a_{max} is obtained when $c = C(a_{max})$. Finally, EFTNAS assembles the search space, $\mathcal{A}$, by combining all the subnetwork configurations in $\mathcal{B}$. Often $N <= 5$ since we can derive a rich search space with just a few subnetwork configurations. N must be at least two since we need at least the subnetwork configurations for the upper and lower bounds of the search space.

An important benefit of using first-order weight importance information when designing the NAS search space is that EFTNAS can effectively set the architectural lower and upper bounds of the search space based on the performance objectives, e.g., the desired latency or GFLOPs ranges. A better-designed and smaller search space reduces the potential interference of subnetworks in regions with an associated performance outside of the desired performance target.

3.3. Transformer-based Super-Network

Super-network Generation Given the search space obtained by Algorithm 1, EFTNAS generates the super-network. The starting point is a transformer-based pre-trained model, e.g., BERT (Devlin et al., 2019). To generate a weight-sharing super-network, i.e., the abstraction that enables the *activation* of smaller subnetworks from a single data structure, EFTNAS enables *elasticity* at selected multi-head attention and intermediate layers of the subsequent feed-forward networks. In the case of the multi-head attention layer, EFTNAS allows the super-network to activate subnetworks with a different number of heads, as illustrated in Figure 2, based on the search space design discussed in the previous section. In the case of the *intermediate* layers of the feed-forward network (FFN) after the attention mechanism, EFTNAS enables variable

Algorithm 1: Automated Generation of the NAS Search Space

Input: Base model, m
Input: Desired minimum subnet computational complexity, $C(a_{min})$
Input: Desired maximum subnet computational complexity, $C(a_{max})$
Input: Number of configurations per layer, N
Output: Search space, $\mathcal{A}$

/* Obtain importance score $\mathbf{S}_m$ (Equation 3) for all elastic layer in m. */
$\mathbf{S}_m \leftarrow$ Score(m)
$step \leftarrow (C(a_{max}) - C(a_{min}))/(N-1)$
$\mathcal{B} \leftarrow \emptyset$
for $c \leftarrow C(a_{min})$ **to** $C(a_{max})$ **by** $step$ **do**
    /* Obtain a new *active* subnetwork configuration by searching for threshold τ to obtain a binary mask $\mathbf{M}_m$ for m, s.t., the associated subnetwork configuration, a has complexity c. $\mathbf{M}_m = 1(\mathbf{S}_m > \tau)$ */
    $\tau \leftarrow$ BinarySearch$(\mathbf{S_m}, c)$
    $a \leftarrow$ SubnetConfig$(\mathbf{M_m}, \tau)$
    $\mathcal{B} \leftarrow \mathcal{B} \cup a$
end for
/* Define search space from boundaries and configurations stored in $\mathcal{B}$ */
$\mathcal{A} \leftarrow$ Combine$(\mathcal{B})$
return $\mathcal{A}$

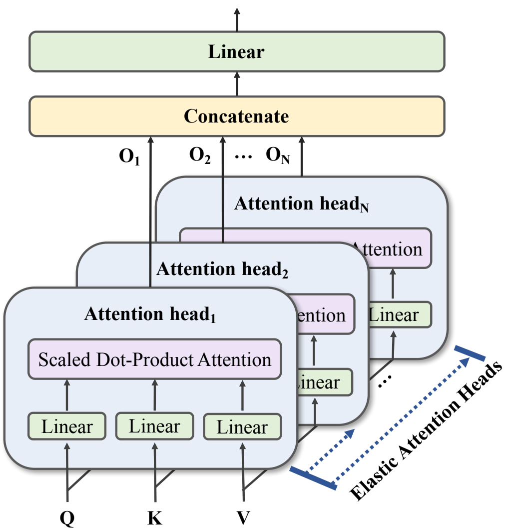

Figure 2: Elastic Number of Attention Heads.

width configurations, as illustrated in Figure 1.

EFTNAS allows a_{max} (the maximal subnetwork) to be different in its architectural configuration than the base pre-trained model, m, used for generating the super-network. That is, EFTNAS has two options for the upper-end configuration of the search space: (i) $config(a_{max}) \leftarrow config(m)$ s.t., at initialization, $Cost(m, D_{val}) \cong Cost(a_{max}, D_{val})$, both the pre-trained model, m and the maximal subnetwork, a_{max} will result in a similar performance on a validation set, D_{val}. (ii) $C(a_{max}) < C(m)$, resulting in an architectural configuration of a_{max} smaller than the configuration of m. In this latter case, we expect a_{max} to have a strong initialization since, as described next, this subnetwork shares the most important weights from the pre-trained model.

First-Order Weight-Reordering Before training the super-network, a standard step in weight-sharing super-networks is to reorder the weights inherited from a previous training stage or the pre-trained model used to generate the super-network. EFTNAS goes beyond zeroth-order weight importance approaches used previously in NAS to re-order the super-network's weights, using the first-order importance score, $\mathbf{S}$ (Equation 3). At each elastic linear layer (of the maximal architecture configuration), its weights tensor, $\boldsymbol{W}$, and its corresponding score tensor, $\mathbf{S}$, have the same shape, allowing us to calculate the mean of the values in each column in $\mathbf{S}$ to sort $\boldsymbol{W}$'s columns. As illustrated in Figure 1, we compute importance scores $\mathbf{S}_q, \mathbf{S}_v$, and $\mathbf{S}_k$ for each attention head. These scores allow EFTNAS to reorder the heads within the multi-head attention layer. In the case of the feed-forward network that follows multi-head attention in the transformer block, we obtain the score $\mathbf{S}_{inter}$ that contains the importance of the channels in this layer.

A particular consideration has to be made in the case of the $\mathbf{Q}$ and $\mathbf{K}$ layers since they should apply the same permutation to their weights. EFTNAS uses the mean of the sum of each corresponding column in these two tensors to sort them accordingly. After weight reordering, the pre-trained model, m used to generate the super-network should have approximately similar performance as the weight-reordered model, m_{w_sorted}, i.e., $Cost(m, D_{val}) \cong Cost(m_{w\_sorted}, D_{val})$ on the same validation data, D_{val}. Using the first-order importance score, $\mathbf{S}$, to reorder the weights of the super-network gives EFTNAS an additional boost in the performance of the Pareto frontier of subnetworks (as shown in the results of Section 4).

Knowledge Distillation To further boost the performance of smaller subnetworks, EFTNAS trains the super-network with the supervision of the pre-trained model that was used to generate the super-network, and with the loss following (Lagunas et al., 2021), i.e.,

$$\mathcal{L} = \alpha_{KD}\mathcal{L}_{KD} + \alpha_{ce}\mathcal{L}_{ce}, \tag{4}$$

$$\mathcal{L}_{KD} = T^2 \sum_i p_i^t(T) \log \frac{p_i^t(T)}{p_i^s(T)} \tag{5}$$

$$p_i^k(T) = \frac{\exp(z_i^k/T)}{\sum_{j=1}^{K}(z_j^k/T)}, \tag{6}$$

where T is a temperature hyperparameter, α_{ce} is the scaling factor for the cross-entropy loss, and α_{KD} is the scaling factor for the distillation loss. $p_i^t(T)$ and $p_i^s(T)$ denote the output probability vector of teacher and student, respectively. z_j^k is the k-th value. K represents the number of classes. Empirically, we have determined that this formulation of knowledge distillation yields good results in EFTNAS super-networks.

4. Experiments

Setup EFTNAS is implemented on top of OpenVINO's Neural Network Compression Framework (NNCF)[1] and its BootstrapNAS solution (Muñoz et al., 2022), benefiting from its module wrapping functionality. We also patch the Transformers[2] repository (Wolf et al., 2019) to enable EFTNAS' elasticity controllers to be called by its *Trainers*. We generate fine-tuned transformer-based super-networks for natural language processing (NLP) tasks using the General Language Understanding Evaluation (GLUE) benchmark (Wang et al., 2019) and the Stanford Question Answering Dataset (SQuAD) (Rajpurkar et al., 2016). To train the super-networks, we apply the *sandwich rule* (Yu and Huang, 2019). EFTNAS uses AdamW (Loshchilov and Hutter, 2019) as the default optimizer; the batch size varies depending on the dataset/task, i.e., 32 for GLUE, 16 for SQuADv1.1 and SQuADv2.0.Weight decay is set to 0 for BERT. The learning rate scheduler uses *Cosine Annealing*. Learning rates vary depending on tasks/datasets. For GLUE and SQuAD, we use values in a range between 2e-5 and 3e-5. The base model for EFTNAS-S1 subnetworks is BERT-base, and BERT-medium for EFTNAS-S2 subnetworks. The search space has five possible configurations per layer, i.e., N=5 in Algorithm 1. The step in computational complexity is equal to the range of computational complexity divided by N-1. The NLP tasks use 3 to 15 epochs to compute the importance score, S, and 4 to 20 epochs to train the super-network. We use the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) (Deb et al., 2002) with 1000 evaluations to obtain the Pareto frontier of high-performing subnetworks. The population size is 40 subnetwork configurations. Figure 3 shows examples of search progression on several super-networks. **We report the performance of the discovered subnetwork without any additional fine-tuning after being extracted from the Pareto front.**

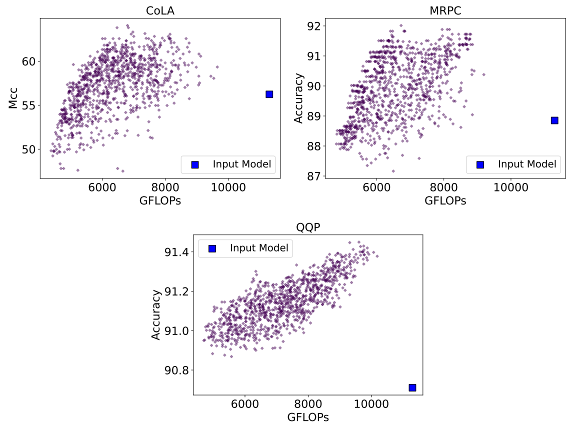

Figure 3: Examples of the search progression using NSGA-II on several EFTNAS super-networks fine-tuned for tasks in the GLUE benchmark. We show 1000 subnetwork configurations sampled for each super-network. Many subnetworks outperform the input base model in efficiency and accuracy.

For a downstream task t, EFTNAS uses Algorithm 2 to discover the best subnetwork that achieves the required performance target, e.g., computational complexity.

4.1. General Language Understanding Evaluation (GLUE) Benchmark

As shown in Table 1, EFTNAS' subnetworks often outperform other approaches in the comparison, resulting in the best average on the GLUE benchmark for the development set and a competitive average on the test set. We compare EFTNAS' subnetworks to DistilBERT (Sanh et al., 2019), TinyBERT (Jiao et al., 2020), MiniLM (Wang et al., 2020b, 2021), AutoDistil (Xu et al., 2022), and NAS-BERT (Xu et al., 2021). Figure 4 illustrates the architectures discovered by EFTNAS-S1 for each of the tasks. Figure 5 shows the search space generated for each task of the GLUE benchmark used to discover the EFTNAS-S1 subnetwork. Each search space is obtained using the proposed approach in Algorithm 1, which derives possible subnetwork configurations based on the desired computational complexity of a few selected subnetworks. We use a maximum of five possible configurations at each layer.

[1] https://github.com/openvinotoolkit/nncf
[2] https://github.com/huggingface

Model	GFLOPs	GLUE Avg.	MNLI-m	QNLI	QQP	SST-2	CoLA	MRPC	RTE
Develoment Set									
$\mathrm{BERT}_{\mathrm{base}}$ (teacher)	11.2	83.3	84.7	91.8	91.0	93.2	59.6	90.4	72.5
$\mathrm{DistilBERT}_6$	5.7	78.6	82.2	89.2	88.5	91.3	51.3	87.5	59.9
$\mathrm{TinyBERT}_6$	5.7	81.9	84.5	91.1	91.1	93.0	54.0	90.6	73.4
MiniLM	5.7	81.0	84.0	91.0	91.0	92.0	49.2	88.4	71.5
$\mathrm{MiniLMv2}(6 \times 768)$	5.7	81.7	84.2	90.8	91.1	92.4	52.5	88.9	72.1
EFTNAS-S1 (Ours)	5.7	**82.9**	**84.6**	90.8	**91.2**	**93.5**	**60.6**	**90.8**	69.0
$\mathrm{NAS{-}BERT}_{10} + \mathrm{KD}$	2.3	74.2	76.4	86.3	88.5	88.6	34.0	79.1	66.6
$\mathrm{AutoDistil}_{\mathrm{Proxy\_S}}$	2.0	79.9	83.2	90.0	90.6	90.1	48.3	88.3	69.4
$\mathrm{AutoDistil}_{\mathrm{Agnostic}}$	2.1	79.6	82.8	89.9	90.8	90.6	47.1	87.3	69.0
EFTNAS-S2 (Ours)	2.2	**80.5**	82.3	88.6	90.4	**91.2**	**52.1**	**90.1**	69.0
Test Set									
$\mathrm{BERT}_{\mathrm{base}}$ (teacher)	11.2	78.2	84.6	90.5	71.2	93.5	52.1	88.9	66.4
$\mathrm{DistilBERT}_6$	5.7	76.8	82.6	88.9	70.1	92.5	49.0	86.9	58.4
$\mathrm{TinyBERT}_6\dagger$	5.7	79.4	84.6	90.4	71.6	93.1	51.1	87.3	70.0
$\mathrm{MiniLMv2}(6 \times 768)$	5.7	77.5	83.8	90.2	70.9	92.9	46.6	89.1	69.2
EFTNAS-S1 (Ours)	5.7	77.7	83.7	89.9	**71.8**	**93.4**	**52.6**	87.6	65.0
EFTNAS-S2 (Ours)	2.2	75.2	82.0	87.8	70.6	91.4	44.5	86.1	64.0

Table 1: Performance comparison on the development and test sets of the GLUE benchmark. We report Matthews' correlation coefficient for CoLA and accuracy (%) for the other tasks. † means using data augmentation.

MNLI	768	3044	256	2465	256	1891	320	1877	640	1825	384	1790	576	1678	256	1544	256	1223	256	1277	192	345	256	213
QNLI	576	1708	448	1066	320	1126	768	1102	640	1067	512	2233	448	1048	448	1670	512	3072	448	772	384	609	576	205
QQP	384	3072	768	1666	320	1787	192	1791	256	1772	256	1751	768	1709	384	2980	192	1320	192	762	192	348	512	115
SST-2	320	1585	256	1570	256	1775	256	1717	384	1679	768	2215	768	2786	320	2114	256	1188	192	930	128	1501	128	654
CoLA	512	949	704	959	320	2733	448	1096	640	2771	768	1774	768	1719	640	1014	576	670	256	436	384	348	320	370
MRPC	768	2120	704	2097	768	928	768	1494	768	3072	768	787	640	672	512	579	576	409	320	291	384	678	192	308
RTE	576	608	576	3072	704	589	768	542	704	576	768	589	768	3072	576	537	704	562	768	453	512	376	640	424

MHA Block **FFN Block**

Figure 4: Configurations for the architectures of each subnetwork discovered by EFTNAS-S1 for each task in the GLUE benchmark. Each number represents the width of the module at that position in the network.

4.2. The Stanford Question Answering Dataset (SQuAD)

As summarized in Table 2, EFTNAS outperforms other approaches and discovers a subnetwork, EFTNAS-S1, with a higher F_1-score for both SQuADv1.1 and SQuADv2.0. We report the number of parameters to compare with other approaches with similar model sizes. We also include the performance of EFTNAS-S2, a significantly smaller subnetwork with a minor drop in the F_1-score.

4.3. Ablation Study: Weight Reordering Strategies

To better understand the importance of the weight reordering strategy when training a super-network, Figure 6 compares the Pareto frontiers obtained after searching on three different super-networks fine-tuned on four downstream tasks from the GLUE benchmark. As the figure shows, using the first-order importance score, $\mathbf{S}$, for weight reordering the weights of the super-network results in better Pareto frontiers. In contrast, the L1-norm (as used in other NAS approaches) tends to degrade the performance of the super-network, performing worse than without weight reordering in some cases.

4.4. Ablation Study: Varying the Number of Possible Configurations for Each Layer

In the main experiments on GLUE (Table 1), EFTNAS generates search spaces with a maximum of five possible width configurations for each layer. Table 3 describes the effects of using a different value for the number of possible configurations for each layer. We experiment using two datasets of the GLUE benchmark, i.e., The Multi-Genre Natural Language Inference (MNLI)(Williams et al., 2018) dataset and the Recognizing Textual Entailment (RTE) dataset (Dagan et al., 2005; Bar-Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009). In both cases, increasing the complexity of the search space results in subnetworks with improved performance: EFTNAS-S1 accuracy in MNLI's increases from 84.6 to 84.8 and from 69.0 to 70.4 in RTE. As future work, we are interested in having an in-depth investigation of the trade-off between search space complexity and the efficiency of the NAS solution.

5. Conclusion

Weight-sharing neural architecture search (NAS) super-networks have proven effective at model com-

MNLI	768, 384, 256	3072, 3044, 2448, 1542	768, 320, 256	3072, 3032, 2465, 1611	768, 384, 256, 192	3072, 3038, 2606, 1891	768, 512, 320	3072, 3026, 2589, 1877	768, 640, 192	3072, 3024, 2564, 1825	768, 576, 384	3072, 2994, 2527, 1790
QNLI	768, 640, 576, 512	3072, 2810, 2272, 1708, 1104	768, 448, 320	3072, 2787, 2272, 1716, 1066	768, 512, 384, 320	3072, 2806, 2338, 1776, 1126	768, 704, 576	3072, 2810, 2281, 1740, 1102	768, 640, 576	3072, 2818, 2310, 1714, 1067	768, 704, 512	3072, 2794, 2233, 1637, 1023
QQP	768, 448, 384, 320	3072, 3018, 2460, 1675	768, 256	3072, 3024, 2488, 1666	768, 448, 320	3072, 3020, 2530, 1787	768, 384, 256, 192	3072, 3007, 2549, 1791	768, 512, 256	3072, 3018, 2543, 1772	768, 512, 256	3072, 2982, 2500, 1751
SST-2	768, 576, 384, 320	3072, 3040, 2727, 2227, 1585	768, 384, 256	3072, 3036, 2712, 2190, 1570	768, 384, 320, 256	3072, 3046, 2795, 2362, 1775	768, 512, 384, 256, 192	3072, 3042, 2791, 2314, 1717	768, 704, 576, 384, 256	3072, 3043, 2764, 2296, 1679	768, 576, 384, 192	3072, 3038, 2768, 2215, 1580
CoLA	768, 704, 576, 512	3072, 2692, 2134, 1588, 949	768, 704, 448, 320, 256	3072, 2685, 2171, 1609, 959	768, 640, 448, 320	3072, 2733, 2275, 1762, 1110	768, 640, 512, 448	3072, 2788, 2312, 1739, 1096	768, 704, 640	3072, 2771, 2316, 1788, 1158	768, 640	3072, 2797, 2312, 1774, 1062
MRPC	768	3072, 2618, 2120, 1524, 995	768, 704	3072, 2621, 2097, 1520, 930	768	3072, 2632, 2117, 1526, 928	768	3072, 2664, 2127, 1494, 931	768	3072, 2645, 2139, 1456, 872	768, 704, 640	3072, 2574, 2019, 1362, 787
RTE	768, 576	3072, 2400, 1773, 1159, 608	768, 576, 448	3072, 2426, 1728, 1119, 571	768, 704, 576	3072, 2419, 1755, 1155, 589	768	3072, 2417, 1742, 1108, 542	768, 704	3072, 2449, 1748, 1124, 576	768, 704	3072, 2446, 1768, 1100, 589

MNLI	768, 576, 128	3072, 3002, 2438, 1678	768, 512, 256	3072, 2985, 2318, 1544	768, 640, 256	3072, 2969, 2035, 1223	768, 320, 256	3072, 2664, 1277, 628	768, 192	3072, 1988, 692, 345	768, 256	3072, 1040, 408, 213
QNLI	768, 704, 576, 448	3072, 2773, 2212, 1642, 1048	768, 512, 448	3072, 2760, 2205, 1670, 1061	768, 576, 512, 448	3072, 2690, 2123, 1548, 984	768, 448, 320	3072, 2568, 1925, 1367, 772	768, 512, 384	3072, 2402, 1751, 1132, 609	768, 640, 576, 512, 448	3072, 2064, 1194, 584, 205
QQP	768, 448, 256, 192	3072, 2974, 2452, 1709	768, 384, 256	3072, 2980, 2403, 1590	768, 384, 192, 64	3072, 2936, 2100, 1320	768, 512, 192	3072, 2721, 1524, 762	768, 320, 192	3072, 1957, 756, 348	768, 512	3072, 483, 197, 115
SST-2	768, 704, 384, 192, 128	3072, 3032, 2786, 2268, 1621	768, 512, 384, 320	3072, 3034, 2694, 2114, 1406	768, 448, 256	3072, 2994, 2506, 1896, 1188	768, 320, 256, 192	3072, 2924, 2236, 1585, 930	768, 128	3072, 2868, 2158, 1501, 828	768, 128, 64	3072, 2812, 1966, 1301, 654
CoLA	768	3072, 2797, 2297, 1719, 1028	768, 640, 576	3072, 2762, 2245, 1654, 1014	768, 576, 512, 448	3072, 2487, 1863, 1261, 670	768, 320, 256	3072, 2187, 1469, 928, 436	768, 384	3072, 2037, 1360, 814, 348	768, 384, 320	3072, 1991, 1322, 799, 370
MRPC	768, 704, 640	3072, 2561, 1955, 1245, 672	768, 704, 640, 512	3072, 2494, 1791, 1117, 579	768, 576, 512	3072, 2391, 1627, 866, 409	768, 640, 512, 320	3072, 2153, 1395, 692, 291	768, 576, 448, 384, 192	3072, 2070, 1324, 678, 300	768, 704, 512, 192	3072, 1925, 1206, 642, 308
RTE	768	3072, 2409, 1730, 1078, 568	768, 704, 640, 576	3072, 2426, 1739, 1054, 537	768, 704	3072, 2407, 1670, 1080, 562	768, 704	3072, 2376, 1633, 1000, 453	768, 640, 512	3072, 2247, 1497, 840, 376	768, 704, 640	3072, 1602, 842, 424, 147

MHA Block FFN Block

Figure 5: Search spaces used to discover EFTNAS-S1 for each task in the GLUE benchmark. Each set of numbers represents the possible width configurations of the module at that position in the network.

Algorithm 2: Discovery and evaluation of the best subnetwork for a downstream task, t, based on a desired computational complexity.

Input: Pre-trained model m
Input: Downstream task t
Input: Desired minimum subnetwork computational complexity, $C(a_{min})$
Input: Desired maximum subnetwork computational complexity, $C(a_{max})$
Input: Desired computational complexity, C_t
Input: Number of configurations per layer, N
Output: Best subnetwork configuration, a^*, and its performance on task t

```
S_m ← ImportanceScore(m)
/* Obtain search space using Algorithm 1 */
A ← SearchSpace(m, C(a_min), C(a_max), N)
Ω ← GenerateSuperNetwork(m, A)
Ω' ← ReorderWeights(Ω, S_m)
Ω* ← Train(Ω')
a* ← Search(Ω*, C_t)
/* No additional fine-tuning of a* is required. */
return (a*, Eval(a*, t))
```

Model	Param.(M)	SQuADv1.1	SQuADv2.0
$\mathrm{BERT_{base}}$ (teacher)	85	88.2	78.6
$\mathrm{DistilBERT_6}$	42	86.9	-
$\mathrm{TinyBERT_6}$	42	87.5	73.4
$\mathrm{MiniLM_6}$	42	-	76.4
$\mathrm{MiniLMv2}(6 \times 768)$	42	-	76.3
EFTNAS-S1 (ours)	42	**88.7**	**78.0**
EFTNAS-S2 (ours)	16	86.8	72.9

Table 2: Comparison of the number of parameters and the F1-score for EFTNAS' subnetworks and other approaches on the SQuAD dataset.

Number of possible configurations per layer	MNLI	RTE
3	84.6	68.2
5	84.6	69.0
7	**84.8**	**70.4**

Table 3: Increasing the number of possible configurations for each layer during the generation of the search space from 5 to 7 configurations results in better accuracy (%) for the subnetwork configuration constrained to 5.7 GFLOPs, i.e., EFTNAS-S1, the MNLI and RTE datasets. However, larger search spaces significantly impact the cost of super-network training and subnetwork search.

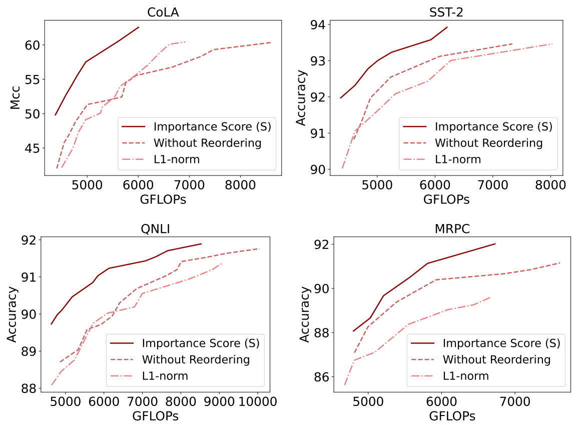

Figure 6: Comparison of the Pareto frontiers of sampled subnetworks from three super-networks trained with different weight-reordering strategies. Using the importance score, $\mathbf{S}$ results in better Pareto frontiers.

pression and specialization. This paper describes EFTNAS, an end-to-end NAS solution that generates robust transformer-based super-networks. EFTNAS incorporates first-order weight-sharing information to automatically generate the NAS search space and reorder the weights of the super-network. The improved search space considers the desired range of computational complexity of the resulting compressed models, improving the efficiency of the NAS training and searching stages since there is no need to explore regions of the search space that might be detrimental to the final objective. The result is a Pareto frontier of several high-performing compressed subnetworks from which we can extract models for several resource budgets. The memory requirements of EFTNAS subnetworks can be further reduced by applying other compression techniques, e.g., quantization, to the resulting subnetworks. We have left these additional optimizations outside the scope of this paper. EFTNAS generates robust transformer-based super-networks. EFTNAS' models and code are available at https://github.com/IntelLabs/Hardware-Aware-Automated-Machine-Learning.

6. Limitations

The methods proposed in this paper have been demonstrated with smaller language models. It is an open research problem how EFTNAS could be efficiently applied to large language models (LLMs). EFTNAS' search stage, in particular, requires significant time and resources. A potential solution for these limitations is parameter-efficient fine-tuning methods (PEFT) that benefit from NAS techniques. For instance, LoNAS (Muñoz et al., 2024b) has attempted to combine NAS and PEFT to search for more efficient LLMs. An improved iteration of LoNAS, Shears (Muñoz et al., 2024a), explores NAS in a space of PEFT adapter configurations using an initial stage that sparsifies and freezes the base model's weights.

7. Ethics Statement

Although large language transformer-based models have achieved significant success lately and are being integrated into many applications, they are prone to output false information, potentially contributing to misinformation. In this paper, we have focused on a particular approach to optimizing language models fine-tuned for a target task so users can deploy them in resource-constrained environments. However, before deployment, we suggest implementing the necessary safeguards to prevent potential harm to others.

Another ethical concern when working with these models is the large number of resources required to train or use them for inference. A positive impact of the approach proposed in this paper is that compressed models have a reduced footprint compared to their based models. There is work to be done by the research community to continue reducing the massive amount of resources that (large) deep learning models tend to consume.

Acknowledgments

We are grateful to Michael Beale from Intel Labs, who helped us set up the infrastructure for sharing our models during the review stage and the final release and guided us through the process of open-sourcing our compressed models. We also thank Vui Seng Chua for his feedback and suggestions regarding neural network pruning. We also thank the anonymous reviewers for their insightful suggestions, which helped us improve the paper. Finally, we thank Jinjie Yuan for helping us set up our models' repository.

8. Bibliographical References

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

Roy Bar-Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second pascal recognising textual entailment challenge. In *Proceedings of the second PASCAL challenges workshop on recognising textual entailment*, volume 6, pages 6–4. Venice.

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. 2018. Understanding and simplifying one-shot architecture search. In *ICML*.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. In *TAC*.

Davis W. Blalock, Jose Javier Gonzalez Ortiz, Jonathan Frankle, and John V. Guttag. 2020. What is the state of neural network pruning? In *MLSys*. mlsys.org.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2020. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*.

Han Cai, Ligeng Zhu, and Song Han. 2019. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*.

Minghao Chen, Houwen Peng, Jianlong Fu, and Haibin Ling. 2021. Autoformer: Searching transformers for visual recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 12270–12280.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine Learning Challenges Workshop*, pages 177–190. Springer.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. 2002. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

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

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and Bill Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9. Association for Computational Linguistics.

Naman Goyal, Jingfei Du, Myle Ott, Giri Anantharaman, and Alexis Conneau. 2021. Larger-scale transformers for multilingual masked language modeling. *arXiv preprint arXiv:2105.00572*.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, pages 544–560. Springer.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. Tinybert: Distilling bert for natural language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4163–4174, Online. Association for Computational Linguistics.

François Lagunas, Ella Charlaix, Victor Sanh, and Alexander Rush. 2021. Block pruning for faster transformers. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10619–10629, Online and

Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yann LeCun, John Denker, and Sara Solla. 1989. Optimal brain damage. In *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. 2018a. Progressive neural architecture search.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018b. Darts: Differentiable architecture search.

Jing Liu, Jianfei Cai, and Bohan Zhuang. 2022. Focusformer: Focusing on what we need via architecture sampler.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Roberta: A robustly optimized bert pretraining approach.

Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. 2021. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

J. Pablo Muñoz, Nikolay Lyalyushkin, Yash Akhauri, Anastasia Senina, Alexander Kozlov, and Nilesh Jain. 2022. Enabling nas with automated supernetwork generation. In *Practical Deep Learning in the Wild, AAAI*.

J. Pablo Muñoz, Jinjie Yuan, and Nilesh Jain. 2024a. Shears: Unstructured sparsity with neural low-rank adapter search. Accessed: 2024-03-05.

J. Pablo Muñoz, Jinjie Yuan, Yi Zheng, and Nilesh Jain. 2024b. Lonas: Elastic low-rank adapters for efficient large language models. In *The 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing.

Michael Poli, Stefano Massaroli, Eric Q. Nguyen, Daniel Y. Fu, Tri Dao, Stephen A. Baccus, Yoshua Bengio, Stefano Ermon, and Christopher Ré. 2023. Hyena hierarchy: Towards larger convolutional language models. *ArXiv*, abs/2302.10866.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Victor Sanh, Thomas Wolf, and Alexander M. Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2022. Efficient transformers: A survey. *ACM Comput. Surv.*, 55(6).

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou. 2021. Training data-efficient image transformers and distillation through attention. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In the Proceedings of ICLR.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020a. Hat: Hardware-aware transformers for efficient natural language processing. In *Annual Conference of the Association for Computational Linguistics*.

Wenhui Wang, Hangbo Bao, Shaohan Huang, Li Dong, and Furu Wei. 2021. MiniLMv2: Multi-head self-attention relation distillation for compressing pretrained transformers. In *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pages 2140–2151, Online. Association for Computational Linguistics.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020b. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA. Curran Associates Inc.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2018. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*.

Colin White, Mahmoud Safari, Rhea Sukthanker, Binxin Ru, Thomas Elsken, Arber Zela, Debadeepta Dey, and Frank Hutter. 2023. Neural architecture search: Insights from 1000 papers.

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics.

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

Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Zhicheng Yan, Masayoshi Tomizuka, Joseph Gonzalez, Kurt Keutzer, and Peter Vajda. 2020. Visual transformers: Token-based image representation and processing for computer vision.

Mengzhou Xia, Zexuan Zhong, and Danqi Chen. 2022. Structured pruning learns compact and accurate models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1513–1528.

Dongkuan Xu, Subhabrata Mukherjee, Xiaodong Liu, Debadeepta Dey, Wenhui Wang, Xiang Zhang, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022. Few-shot task-agnostic neural architecture search for distilling large language models. In *Advances in Neural Information Processing Systems*.

Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. 2021. Nas-bert: Task-agnostic and adaptive-size bert compression with neural architecture search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 1933–1943, New York, NY, USA. Association for Computing Machinery.

Peng Xu, Xiatian Zhu, and David A Clifton. 2023. Multimodal learning with transformers: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Yichun Yin, Cheng Chen, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2021. Autotinybert: Automatic hyper-parameter optimization for efficient pre-trained language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5146–5157, Online. Association for Computational Linguistics.

Jiahui Yu and Thomas S. Huang. 2019. Universally slimmable networks and improved training techniques. *CoRR*, abs/1903.05134.

Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas S. Huang, Xiaodan Song, Ruoming Pang, and Quoc V. Le. 2020. Bignas: Scaling up neural architecture search with big single-stage models. *CoRR*, abs/2003.11142.

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. 2022. Scaling vision transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12104–12113.

Li Zhang, Jiachen Lu, Sixia Zheng, Xinxuan Zhao, Xiatian Zhu, Yanwei Fu, Xiang Tao, and Jianfeng Feng. 2023. Vision transformers: From semantic segmentation to dense prediction. *arXiv*.