Exploring Learngene via Stage-wise Weight Sharing for Initializing Variable-sized Models

Shi-Yu Xia , **Wenxuan Zhu** , **Xu Yang*** and **Xin Geng***
School of Computer Science and Engineering, Southeast University, Nanjing 210096, China
Key Laboratory of New Generation Artificial Intelligence Technology and Its Interdisciplinary Applications (Southeast University), Ministry of Education, China
{shiyu_xia, zhuwx, xuyang_palm, xgeng}@seu.edu.cn

Abstract

In practice, we usually need to build variable-sized models adapting for diverse resource constraints in different application scenarios, where weight initialization is an important step prior to training. The Learngene framework, introduced recently, firstly learns one compact part termed as **learngene** from a large well-trained model, after which learngene is expanded to initialize variable-sized models. In this paper, we start from analysing the importance of guidance for the expansion of well-trained learngene layers, inspiring the design of a simple but highly effective Learngene approach termed SWS (***S**tage-wise **W**eight **S**haring*), where both learngene layers and their learning process critically contribute to providing knowledge and guidance for initializing models at varying scales. Specifically, to learn learngene layers, we build an auxiliary model comprising multiple stages where the layer weights in each stage are shared, after which we train it through distillation. Subsequently, we expand these learngene layers containing *stage information* at their corresponding stage to initialize models of variable depths. Extensive experiments on ImageNet-1K demonstrate that SWS achieves consistent better performance compared to many models trained from scratch, while reducing around **6.6**× total training costs. In some cases, SWS performs better only after **1 epoch** tuning. When initializing variable-sized models adapting for different resource constraints, SWS achieves better results while reducing around **20**× parameters stored to initialize these models and around **10**× pre-training costs, in contrast to the pre-training and fine-tuning approach.

1 Introduction

Vision Transformers (ViTs) have become increasingly popular, showcasing their remarkable performance across a wide range of vision tasks [Dosovitskiy *et al.*, 2021; Liu *et al.*, 2021; Wang *et al.*, 2022b; Oquab *et al.*, 2023; Yan *et al.*, 2023]. In practical deployment, it is often necessary to train models of *various scales* to flexibly accommodate different resource constraints. These constraints may exhibit significant diversity, such as mobile devices with limited available resources and computing centers with substantial computational capabilities. Clearly, training each target model from scratch provides a straightforward solution, where weight initialization is a crucial step prior to training which aids in model convergence and affects the final quality of the trained model [Glorot and Bengio, 2010; He *et al.*, 2015; Arpit *et al.*, 2019; Huang *et al.*, 2020].

Nowadays, a variety of large-scale pretrained models, developed by the research and industry community, are readily available to transfer and finetune the learned weights for diverse downstream tasks [Radford *et al.*, 2021; He *et al.*, 2022; Touvron *et al.*, 2023; Oquab *et al.*, 2023]. However, such scheme needs to reuse the original whole pretrained model parameters every time facing different downstream tasks regardless of the available resources. Unfortunately, for many pretrained model families (MAE [He *et al.*, 2022]), even the smallest model (86M ViT-Base [Dosovitskiy *et al.*, 2021]) can be considered extremely large for some resource-constrained settings. To tackle this, developers would have to first pretrain target model to meet certain resource demand, which is time-consuming, computationally expensive and lacks the flexibility to initialize models of *varying scales*. Thus, how to flexibly initialize diverse models to satisfy different resource constraints arises as an important research question.

Recently, [Wang *et al.*, 2023] proposes a novel learning paradigm known as *Learngene* to achieve this goal. As shown in Fig. 1(a), Learngene firstly learns one compact part termed as **learngene**, which contains generalizable knowledge, from a large well-trained network termed as ancestry model (Ans-Net). Subsequently, learngene is expanded to initialize variable-sized descendant models (Des-Net), after which they undergo normal fine-tuning. Based on the gradient information of Ans-Net, Vanilla-LG [Wang *et al.*, 2022a] extracts a few high-level layers as learngene and combines them with randomly-initialized layers to build Des-Nets. TLEG [Xia *et al.*, 2024] extracts two layers as learngene which is linearly expanded to build varying Des-Nets. LearngenePool [Shi *et al.*, 2024] distills one large model into multiple small ones whose layers are used as learngene instances and then stitches them to build Des-Nets. However, there exist several limitations in previous works. Firstly, the extracted learngene

*Co-corresponding author.

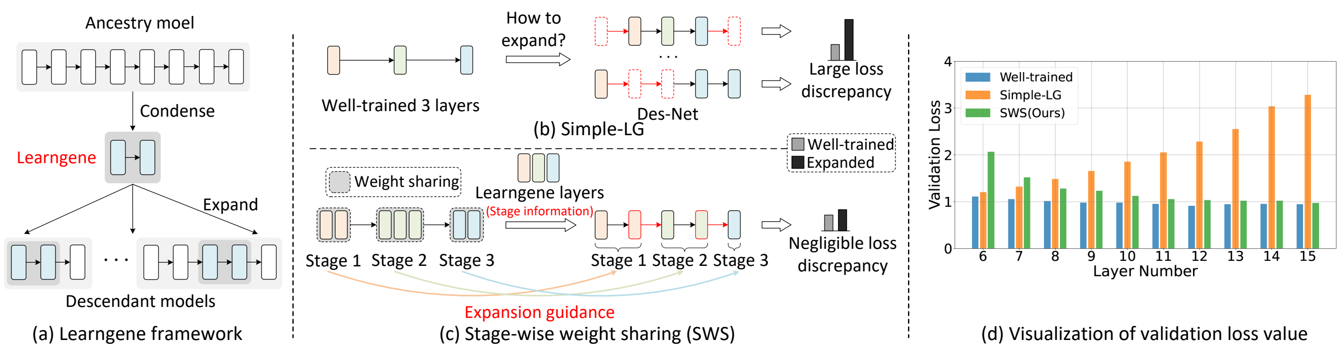


Figure 1: (a) Learngene. (b) Simple-LG and (c) SWS, here we take 3-layer learngenes as an example. (d) Visualization of validation loss value.

itself only contains well-trained parameters but lacks crucial knowledge essential for the subsequent expansion process. Secondly, the learngene learning process provides insufficient guidance about how to effectively and conveniently expand the learngene, thereby constraining its potential for initialization. Thirdly, empirical performance still lags far behind the pre-training and fine-tuning approach.

To validate the importance of such knowledge and guidance, we study from an idea termed Simple-LG: we first train a vanilla 5-layer model whose layers compose learngene layers, and then expand its layer to initialize Des-Nets of required depth, as shown in Fig. 1(b). Obviously, neither the well-trained layers nor the learning process of the 5-layer model possess knowledge or guidance for the subsequent expansion process. As a result, this strategy brings severe performance degradation compared to well-trained ones. For example, it leads to a 27.1% accuracy degradation of initialized 12-layer Des-Net in ImageNet-1K [Deng *et al.*, 2009] classification without any fine-tuning. Moreover, we display the validation loss value of these initialized models in Fig. 1(d), which delivers two important messages: 1) These initialized models are not fine-tuned, yet they attain meaningful loss value, which demonstrates the potential of expanding learngene layers to initialize models. 2) With the increasing layer number of models, *i.e.*, the increasing number of expanding layers, the discrepancy between the validation loss of initialized models and that of well-trained ones becomes larger.

Upon closer observation, if we treat each learngene layer as an individual stage, we can *recursively update* each layer at its stage, which equals to sharing weights across multiple layers within each stage throughout the training, shown in left of Fig. 1(c). In this way, we seamlessly integrate *stage information* into each learned learngene layer, *i.e.*, learngene layer actually contains knowledge of multiple weight-shared layers within its stage. Moreover, we could emulate the learngene expansion process via adding weight-shared layers within each stage during learngene training, thereby providing clear guidance on how to expand, namely, expanding the weight-shared layers at its corresponding stage, shown in Fig. 1(c). Both stage information and expansion guidance are necessary: 1) expanding learngene layers which lacks stage information destroys the intrinsic layer connection (See Simple-LG). 2) Without expansion guidance, the position of expanded layers remains uncertain. Based on this insight, we present ***Stage-wise Weight Sharing*** (**SWS**), a simple but effective Learngene approach for efficient model initialization. SWS divides one Transformer into multiple stages and shares the layer weights within each stage during the training process. Specifically, we design and train an auxiliary model (Aux-Net) whose layer weights are shared via SWS to obtain learngene layers. Then we can expand these layers at their corresponding stage to initialize variable-sized Des-Nets. As shown in Fig. 1(d), we observe that the validation loss of models initialized by SWS can be significantly reduced with the increasing layer numbers.

We systematically investigate the design of weight sharing and the initialization strategy. With extensive experiments, we show the superiority of SWS: 1) Compared to training from scratch, SWS achieves better performance with much less training efforts on ImageNet-1K. Take Des-B as an example, SWS performs better while reducing around **6.6**$\times$ total training costs. 2) When transferring to downstream classification datasets, SWS surpasses existing Learngene methods by a large margin, *e.g.*, **+5.6%** on Cars-196. 3) When directly evaluating on ImageNet-1K without any tuning after initialization, SWS outperforms existing initialization methods by a large margin, *e.g.*, **+9.4%** with Des-B (86M). 4) When building variable-sized models, SWS achieves better results while reducing around **20**$\times$ parameters stored to initialize and around **10**$\times$ pre-training costs, in contrast to the pre-training and fine-tuning. Our main **contributions** are summarized as:

- We propose a simple but effective Learngene approach termed SWS for efficient model initialization, which is the first work to systematically explore the potential of weight sharing for initializing variable-sized models.
- We present the design of weight sharing and the initialization strategy, which firstly highlights the importance of stage information and expansion guidance.
- Extensive experiments demonstrate the effectiveness and efficiency of SWS, *e.g.*, compared to training from scratch, training with compact learngenes can achieve better performance while reducing huge training costs.

2 Related Work

2.1 Weight Initialization

Weight initialization is a pivotal step prior to training one model and crucially affects the model performance [Glorot

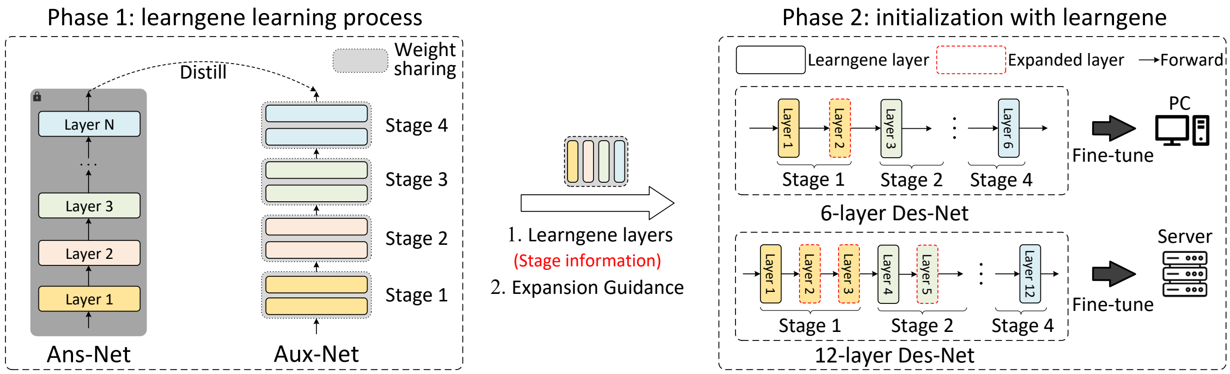


Figure 2: In the first phase, we build an auxiliary model comprising multiple stages. The layer weights in each stage are shared. Note that the number of layers in each stage and the number of stages are both configurable. Then we train it via distillation. After the learngene learning process, learngene layers containing stage information and expansion guidance are adopted to initialize descendant models of variable depths in the second phase. Finally, these models are fine-tuned normally and deployed to practical scenarios with diverse resource constraints.

and Bengio, 2010; He *et al.*, 2015; Mishkin and Matas, 2015; Arpit *et al.*, 2019; Huang *et al.*, 2020]. Proper initialization aids in model convergence and training efficiency [LeCun *et al.*, 2002], while arbitrary initialization may hinder the training process [Mishkin and Matas, 2015]. Comprehensive initialization strategies have been proposed, such as default initialization from Timm library [Paszke *et al.*, 2019], Xavier initialization [Glorot and Bengio, 2010] and Kaiming initialization [He *et al.*, 2015]. Nowadays, a plethora of pretrained models are readily accessible, offering an excellent initialization for fine-tuning models across a range of downstream tasks [Bao *et al.*, 2022; Oquab *et al.*, 2023]. However, this approach needs to reuse the *entire model* for each distinct downstream task, irrespective of the available resources. Furthermore, we need to pre-train again in instances where a pretrained model of the required size is unavailable, which is extremely time-consuming and computationally expensive. Recently, [Xu *et al.*, 2023; Samragh *et al.*, 2023] propose to initialize small models with a larger pretrained model. By contrast, we seek to train compact learngenes *once* via stage-wise weight sharing and then we can initialize variable-sized models.

2.2 Learngene

Learngene proposes to firstly learn a compact part, referred to as learngene, from a large well-trained model termed as ancestry model (Ans-Net) [Wang *et al.*, 2023; Feng *et al.*, 2024]. Subsequently, learngene is expanded to initialize variable-sized descendant models (Des-Net), after which they undergo normal fine-tuning. Vanilla-LG [Wang *et al.*, 2022a] extracts a few high-level layers as learngene and combines them with randomly initialized layers to build Des-Nets. TLEG [Xia *et al.*, 2024] linearly expands learngene which consists of two layers to initialize Des-Nets of varying scales. LearngenePool [Shi *et al.*, 2024] distills one pretrained model into multiple small ones whose layers are used as learngene instances, after which they are stitched to build Des-Nets. In contrast, we firstly investigate integration of stage information into learned learngenes and explore obtaining useful guidance about how to expand learngenes from the learngene learning process, thus better initializing Des-Nets.

2.3 Weight Sharing

Weight sharing is a parameter-efficient model compression strategy [Dabre and Fujita, 2019; Lan *et al.*, 2020; Takase and Kiyono, 2021; Zhang *et al.*, 2022; DING *et al.*, 2023], which effectively alleviates over-parameterization problem [Bai *et al.*, 2019; Kovaleva *et al.*, 2019] in large pretrained Transformers [Devlin *et al.*, 2018]. Different from existing works, we seek to initialize variable-sized models using learngene learned via stage-wise weight sharing, which to our knowledge remains unexplored in the literature.

3 Approach

Fig. 2 depicts the overall pipeline of SWS. In phase 1, we design an auxiliary Transformer model (Aux-Net) comprising several distinct stages, where the layer weights in each stage are shared. We train the Aux-Net through distilling from the ancestry model (Ans-Net) to help learn learngene layers and note that the weights of only one layer is trained in each stage due to the sharing mechanism. In phase 2, the well-trained learngene layers containing *stage information* as well as *expansion guidance* are adopted to initialize descendant models (Des-Net) of variable depths. Finally, these Des-Nets are fine-tuned normally without the restriction of stage-wise weight sharing. Next, we firstly introduce some preliminaries.

3.1 Preliminaries

Thanks to the modular design of modern vision transformer (ViT) [Dosovitskiy *et al.*, 2021; Touvron *et al.*, 2021], a typical ViT of L layers equipped with parameters θ can be defined as a composition of functions: $f_\theta = f_L \circ \cdots \circ f_1$, where $f_\theta : \mathcal{X} \to \mathcal{Y}$ transforms the inputs in an input space $\mathcal{X}$ to the output space $\mathcal{Y}$, f_i means the function of the i-th layer and $\circ$ indicates the composition. Each layer contains Multi-head Self-Attention (MSA) and Multi-Layer Perceptron (MLP) block, where Layer Normalization (LN) [Ba *et al.*, 2016] and residual connections are used before and after each block. The basic idea of weight sharing involves sharing parameters across layers, which is a simple but effective strategy to improve parameter efficiency. For multi-stage weight sharing, weight sharing in the m-th

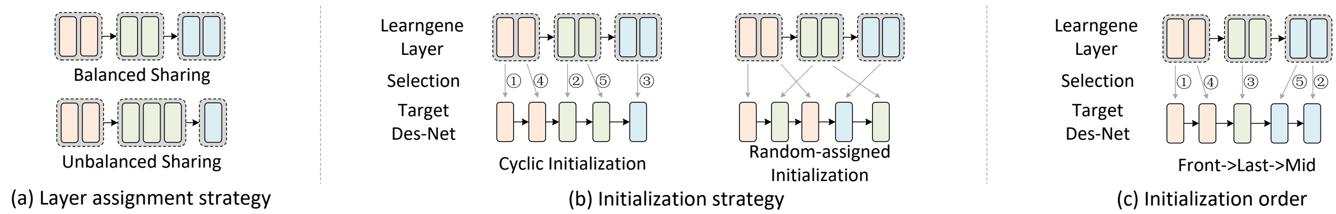


Figure 3: Taking $M = 3$ as an example, we show (a) Layer assignment strategy, (b) Initialization strategy and (c) Initialization order.

stage can be defined as a recursive update of one shared layer:

$$Z_{i+1} = H_m(Z_i, \theta_m), \quad i = 0, 1, ..., L_m - 1, \tag{1}$$

where H_m denotes the function of m-th stage, L_m denotes the number of weight-shared layers in the m-th stage, Z_i denotes the representations in the i-th layer of m-th stage and θ_m represents the parameters of H_m, *i.e.*, shared weights of the L_m layers. Note that the parameters of layers in each stage are shared, *i.e.*, the number of updated parameters in each stage during training equals to that of one layer, but the parameters between stages are not shared. Thus ViT with parameters θ can be further defined as $f_\theta = H_M \circ \cdots \circ H_1$ in multi-stage weight sharing setting, where M denotes the number of stages.

3.2 Stage-wise Weight Sharing of Learngene

As discussed before, we could seamlessly integrate *stage information* into each learned learngene layer via sharing weights across multiple layers within each stage throughout the training. Drawing from this insight, we propose to share the parameters of one learngene layer to form multiple weight-shared layers in each stage. Thus, the number of learngene layers equals to that of stages. By configuring the number of stages M, we can obtain different number of learngenes $\theta_{lg} = \{\theta_1, ..., \theta_M\}$. To ensure clarity, taking the 16-layer ViT-B (114M) [Dosovitskiy *et al.*, 2021] and $M = 5$ as an example, θ_{lg} comprises about 36M parameters which is approximately equivalent to the parameter numbers of five layers and only 36M parameters are updated during the learngene learning process. In the following, we detail the layer assignment strategy and learngene learning process.

Layer Assignment Strategy. Given the number of stages M, there are two options to assign the number of layers within each stage: *Balanced* and *Unbalanced* sharing mechanism. As shown in Fig. 3(a), Balanced sharing ensures uniformity in the layer numbers across most stages, whereas Unbalanced sharing sets uneven layer numbers within most stages. However, as Balanced sharing is more aligned with the existing weight sharing principle [Lan *et al.*, 2020; Takase and Kiyono, 2021], we will show in Section 4.3 that it achieves more stable and better learngenes than Unbalanced sharing. In this case, we take Balanced sharing as the default sharing mechanism in SWS.

Learngene Learning Process. As the learngene layer is the Transformer layer, but some other components like the patch projection and task-specific head are also required to compose a complete Transformer model. Therefore, we also add them to build the Aux-Net, after which we train it through distillation. Specifically, we consider adopting prediction-based distillation [Hinton *et al.*, 2015] to condense knowledge from the Ans-Net, which is achieved by minimizing cross-entropy loss between the probability distributions over the output predictions of the Ans-Net and those of the Aux-Net. Overall, one distillation loss is defined as:

$$\mathcal{L}_d = CE(\phi(p_s/\tau), \phi(p_t/\tau)), \tag{2}$$

where $CE(\cdot,\cdot)$ denotes soft cross-entropy loss, p_t denotes the output logits of the pretrained Ans-Net (*e.g.*, Levit-384 [Graham *et al.*, 2021]), p_s denotes the output logits of the Aux-Net, τ denotes the temperature value of distillation and ϕ denotes the softmax function. Moreover, we can seamlessly incorporate advanced distillation techniques [Zhang *et al.*, 2022; Ren *et al.*, 2023; Ji *et al.*, 2023; Li *et al.*, 2024; Li *et al.*, 2025] into our training process. Besides distillation, we also introduce one classification loss:

$$\mathcal{L}_{cls} = CE(\phi(p_s), y), \tag{3}$$

where y denotes ground-truth label. Therefore, our total training loss is defined as:

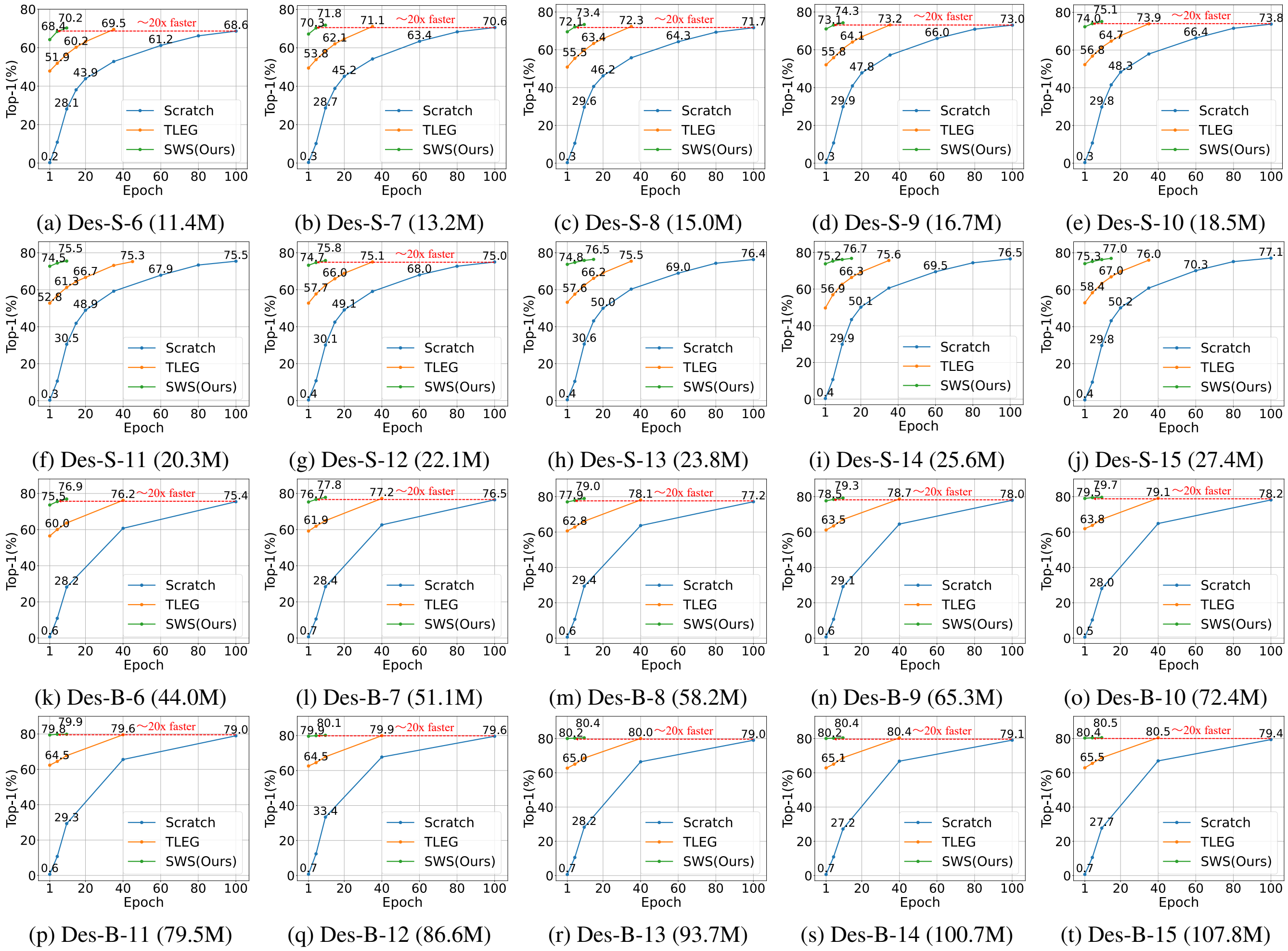
$$\mathcal{L} = (1 - \alpha)\mathcal{L}_{cls} + \alpha\mathcal{L}_d, \tag{4}$$

where α denotes the trade-off. Noteworthy, the weight sharing constraint always exists during training, *i.e.*, although Aux-Net contains L layers, only $\theta_{lg} = \{\theta_1, ..., \theta_M\}$ which contains parameters of M layers are updated.

3.3 Initialization with Learngene

After obtaining learngene which is composed of well-trained $\{\theta_1, ..., \theta_M\}$, we can flexibly initialize Des-Nets of varying depths L^{ds}, fitting diverse resource constraints with much less fine-tuning efforts. In the following, we describe the initialization strategy and the initialization order.

Initialization Strategy. During the learngene learning process, sharing weights in a stage-wise fashion provides important *expansion guidance* for initializing Des-Nets, *i.e.*, expanding the weight-shared layers at its corresponding stage to initialize the target Des-Net. Specifically, we present two initialization strategies: *Cyclic Initialization* and *Random-assigned Initialization*. As shown in Fig. 3(b), we illustrate the initialization process of a 5-layer Des-Net with 3 learngene layers as an example. Cyclic Initialization involves the sequential selection of learngene layers in a specific order to initialize corresponding layers of the Des-Net. In contrast, Random-assigned strategy initializes the Des-Net by randomly selecting learngene layers. However, as Cyclic Initialization is more aligned with the learngene learning process (*i.e.*, keeping



(a) Des-S-6 (11.4M) (b) Des-S-7 (13.2M) (c) Des-S-8 (15.0M) (d) Des-S-9 (16.7M) (e) Des-S-10 (18.5M)
(f) Des-S-11 (20.3M) (g) Des-S-12 (22.1M) (h) Des-S-13 (23.8M) (i) Des-S-14 (25.6M) (j) Des-S-15 (27.4M)
(k) Des-B-6 (44.0M) (l) Des-B-7 (51.1M) (m) Des-B-8 (58.2M) (n) Des-B-9 (65.3M) (o) Des-B-10 (72.4M)
(p) Des-B-11 (79.5M) (q) Des-B-12 (86.6M) (r) Des-B-13 (93.7M) (s) Des-B-14 (100.7M) (t) Des-B-15 (107.8M)

Figure 4: Performance comparisons on ImageNet-1K between several baselines and SWS. Number in bracket of (a)-(t) means Params(M).

the stage order the same as the Aux-Net), we will show in Section 4.3 that it achieves more stable and better performance than Random-assigned initialization. In this case, we take Cyclic Initialization as the default initialization strategy.

Initialization Order. Besides, different initialization orders lead to differences in initialization focus. For example, in the case of "Front-Last-Mid", such initialization focuses more on the front and last part of the Des-Net, as shown in Fig. 3(c). With Cyclic Initialization, different initialization orders bring similar performance of Des-Net. Therefore, we take "Front-Mid-Last" as the default initialization order in SWS.

4 Experiments

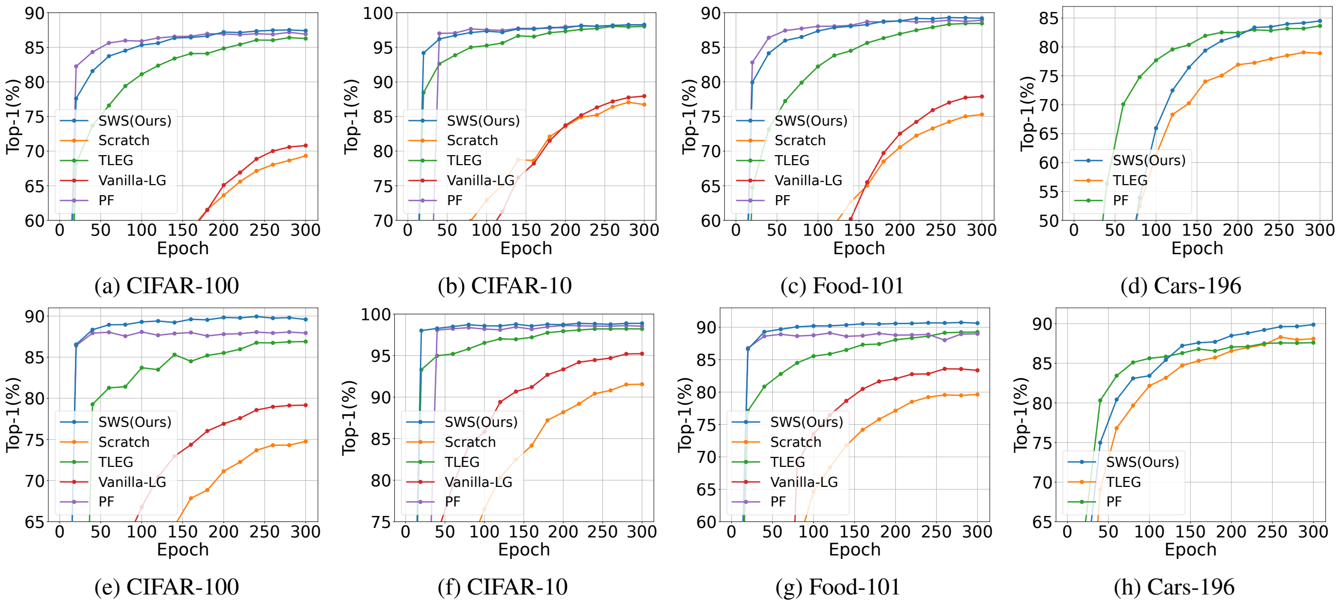
4.1 Experimental Setup

We perform experiments on ImageNet-1K [Deng *et al.*, 2009] and several downstream datasets including CIFAR-10, CIFAR-100 [Krizhevsky *et al.*, 2009], Food-101 [Bossard *et al.*, 2014] and Cars-196 [Krause *et al.*, 2013]. Model performance is measured by Top-1 classification accuracy (Top-1(%)). And we report Params(M) and FLOPs(G) as the number of model parameters and indicators of theoretical complexity of model.

In the first phase, we set two variants of Aux-Net as Aux-S/B where we adopt SWS on DeiT-S/B [Touvron *et al.*, 2021], after which we train Aux-S/B on ImageNet-1K for 150/100 epochs to obtain learngenes, respectively. Specifically, we set 5 stages for Aux-S/B, where the shared layer number in each stage is 3,3,4,3 and 3. We choose Levit-384 [Graham *et al.*, 2021] as the ancestry model. In the second phase, we set two variants of Des-Net as Des-S/B where we change the layer numbers based on DeiT-S/B, *e.g.*, we name the 6-layer Des-S as Des-S-6. Then we initialize Des-S/B with cyclic initialization and fine-tune them for 10 epochs, except that 15 epochs for Des-S-13, Des-S-14 and Des-S-15 for better performance. Source code is available at https://github.com/AlphaXia/SWS.

4.2 Main Results

SWS achieves better performance while reducing huge training efforts in contrast to from scratch training on ImageNet-1K. We report ImageNet-1K classification performance of 20 different Des-Nets in Fig. 4, where "Scratch" denotes training from scratch, "TLEG" denotes linearly expanding learngenes to initialize [Xia *et al.*, 2024]. Compared to Scratch, SWS can achieve better performance and signif-



(a) CIFAR-100 (b) CIFAR-10 (c) Food-101 (d) Cars-196

(e) CIFAR-100 (f) CIFAR-10 (g) Food-101 (h) Cars-196

Figure 5: Performance comparisons on several downstream classification datasets of (a)-(d): Des-S-12 and (e)-(h): Des-B-12.

L^{ds}	Params (M)	FLOPs (G)	Trained (100ep)	IMwLM	Vanilla -LG	TLEG	SWS
6	44.0	8.8	75.4	3.3	0.1	39.9	**59.9**
7	51.1	10.3	76.5	5.8	0.1	60.4	**68.5**
8	58.2	11.7	77.2	12.2	0.1	69.8	**74.4**
9	65.3	13.1	78.0	24.2	0.1	73.8	**76.5**
10	72.4	14.6	78.2	42.2	0.1	75.7	**78.0**
11	79.5	16.0	79.0	59.2	0.1	76.4	**78.9**
12	86.6	17.5	79.6	69.9	0.1	76.6	**79.3**
13	93.7	18.9	79.0	76.2	0.1	76.7	**80.0**
14	100.7	20.4	79.1	78.6	0.1	76.5	**80.1**
15	107.8	21.8	79.4	79.3	0.1	76.0	**80.5**

Table 1: Performance comparisons on ImageNet-1K of Des-B with different layer numbers without any tuning after initialization.

L^{ds}	Params (M)	PF		SWS	
		P-S(M)	Top-1(%)	P-S(M)	Top-1(%)
6	43.4	44.0	87.99		**88.11**
7	50.4	51.1	88.04		**88.75**
8	57.5	58.2	88.35		**89.01**
9	64.6	65.3	88.40		**89.35**
10	71.7	72.4	88.04	37.0	**89.06**
11	78.8	79.5	88.80		**89.21**
12	85.9	86.6	88.47		**89.59**
13	93.0	93.7	88.36		**89.80**
14	100.1	100.7	88.24		**89.28**
15	107.1	107.8	88.34		**89.20**

Table 2: Performance comparisons on CIFAR-100 of Des-B with different layer numbers. For each target model, PF transfers all the pretrained parameters (P-S(M)) to initialize, which totally requires 759.3M for 10 Des-Bs. In contrast, SWS only needs to store 37.0M parameters to initialize each Des-B, which significantly reduces the parameters stored for initialization by 20× (759.3M *vs.* 37.0M).

icantly improve training efficiency. Take 10 Des-Bs as an example, SWS performs better while reducing around **6.6**× total training costs (10×100 epochs *vs.* 100+10×5 epochs), compared to training each Des-B from scratch for 100 epochs. For each Des-B, SWS can reduce around **20**× training costs. In some cases such as from Des-B-10 to Des-B-15, SWS performs better only after **1 epoch** tuning, which demonstrates the effectiveness of initialization via SWS. Compared to TLEG, SWS performs better and further enhances the efficiency. Take 10 Des-Bs as an example, SWS performs better while reducing around **2.5**× total training costs (100+10×40 epochs *vs.* 100+10×10 epochs). In a nutshell, the efficiency of SWS becomes more obvious with the number of Des-Nets increasing as we only need to train learngenes *once*.

When transferring to downstream classification datasets, SWS presents competitive results. We compare SWS against pre-training and fine-tuning (**PF**), Scratch, Vanilla-LG [Wang *et al.*, 2022a] and TLEG [Xia *et al.*, 2024] on 4 classification datasets. As shown in Fig. 5, we observe that SWS consistently outperforms several baselines, which verifies the effectiveness of initializing with learngenes trained via SWS. Take Des-B-12 as an example, SWS consistently outperforms PF by **1.12%**, **0.35%**, **1.65%** and **2.28%** respectively on CIFAR-100, CIFAR-10, Food-101 and Cars-196.

SWS significantly outperforms initialization baselines when directly evaluating on ImageNet-1K without any tuning after initialization. To validate the initialization quality of SWS, we compare SWS against Trained, Vanilla-LG [Wang *et al.*, 2022a], TLEG [Xia *et al.*, 2024] and IMwLM [Xu *et al.*, 2023] on ImageNet-1K, where Trained means models trained from scratch for 100 epochs and IMwLM means initializing

L^{ds}	Params(M)	FLOPs(G)	Simple-LG	SWS
8	58.2	11.7	66.5	**74.4**
9	65.3	13.1	63.4	**76.5**
10	72.4	14.6	59.9	**78.0**
11	79.5	16.0	57.2	**78.9**
12	86.6	17.5	52.5	**79.3**

Table 3: Performance comparisons of Des-Bs on ImageNet-1K initialized from learngenes trained with or without SWS. Simple-LG means initializing from a normally well-trained 5-layer Des-B.

L^{ds}	Params(M)	FLOPs(G)	RI	CI-order1	CI-order2
7	51.1	10.3	50.5	68.5	**70.0**
8	58.2	11.7	47.8	**74.4**	72.6
9	65.3	13.1	71.2	**76.5**	74.2
10	72.4	14.6	70.7	**78.0**	**78.0**
11	79.5	16.0	76.8	**78.9**	**78.9**
12	86.6	17.5	75.2	79.3	**79.9**

Table 4: Performance of Des-Bs on ImageNet-1K under different strategies. "RI" means Random-assigned Initialization. "CI-order1" and "CI-order2" mean Cyclic Initialization with different orders.

small models from a larger model. From Table 1, we observe that SWS significantly outperforms all baselines by a large margin. For example, SWS outperforms IMwLM by $\mathbf{52.3\%}$, $\mathbf{35.8\%}$, $\mathbf{19.7\%}$ and $\mathbf{9.4\%}$ respectively on Des-B-9, Des-B-10, Des-B-11 and Des-B-12. Notably, we also find that directly initializing via SWS *without any tuning* can achieve comparable performance with well-trained models. For example, SWS outperforms Trained by $\mathbf{1.0\%}$, $\mathbf{1.0\%}$ and $\mathbf{1.1\%}$ respectively on Des-B-13, Des-B-14 and Des-B-15, which verifies that the stage-wise information has been preserved to learngenes.

When initializing variable-sized models, SWS notably reduces the parameters stored to initialize compared with pre-training and fine-tuning (PF). We report the results of Des-Bs initialized from learngenes and those initialized from pretrained parameters whose number equals to that of target model. From Table 2, SWS achieves better performance and significantly reduces $\mathbf{20\times}$ (759.3M *vs.* 37.0M) parameters stored to initialize, in constrast to PF. Furthermore, SWS only needs to train learngene *once* while PF requires pretraining each Des-B individually, thereby significantly reducing the pre-training costs. Specifically, SWS reduces pre-training costs by $\mathbf{10\times}$ (10×100 epochs *vs.* 1×100 epochs) compared to PF. It is noteworthy that the efficiency of SWS becomes more obvious with the increasing number of Des-Nets.

4.3 Ablation and Analysis

The effect of stage-wise weight sharing. As shown in Table 3, we observe that SWS significantly outperforms Simple-LG when directly evaluating on ImageNet-1K without any tuning after initialization, which demonstrates the importance of stage information and expansion guidance for initializing Des-Nets. For example, SWS outperforms Simple-LG by $\mathbf{18.1\%}$ and $\mathbf{21.7\%}$ respectively on Des-B-10 and Des-B-11.

Method	Params (M)	FLOPs (G)	Top-1 (%)
Scratch	86.6	17.5	79.6
TLEG [Xia *et al.*, 2024]	15.7	17.6	76.7
Mini-DeiT [Zhang *et al.*, 2022]	44.1	17.5	**80.9**
MS-WS1 (2,2,2,2,2,2)	44.0	17.5	80.7
MS-WS1 (wo dis)	44.0	17.5	77.7
MS-WS2 (1,4,1,1,4,1)	44.0	17.5	80.0
MS-WS3 (1,5,1,4,1)	37.0	17.5	79.4
MS-WS4 (4,1,6,1,4)	37.0	23.3	80.7
MS-WS5 (4,1,5,2,3)	37.0	21.8	fail
MS-WS6 (3,3,4,3,3)	37.0	23.3	**80.9**

Table 5: Performance of Aux-B on ImageNet-1K under different SWS strategies. Number in brackets means the shared layer number in each stage, separated by commas. "wo dis" means training without distillation. Take "MS-WS6 (3,3,4,3,3)" as an example, it means 5 stages where the shared layer number in each stage is 3,3,4,3,3.

Aux-S	CIFAR-100	CIFAR-10	Food-101	Cars-196
76.4 (100)	87.40	98.26	89.19	84.50
78.7 (150)	**88.54**	**98.58**	**89.83**	**87.88**

Table 6: Performance of Des-S-12 with learngenes trained under different epochs. "Aux-S" means the performance of Aux-S on ImageNet-1K. The number in bracket means training epochs.

The effect of different stage-wise weight sharing strategies. We present the performance of Aux-B to show the quality of trained learngenes. From Table 5, we observe that MS-WS6 outperforms MS-WS4 and MS-WS5 fails to converge, which reflects balanced sharing is more stable and better than unbalanced one. Moreover, we find that MS-WS4 outperforms MS-WS3, which demonstrates starting from a weight-sharing stage is better. Also, MS-WS5 achieves comparable performance in contrast to Mini-DeiT [Zhang *et al.*, 2022].

The effect of different initialization strategies. From Table 4, we find that performance of Des-Nets with Cyclic Initialization (CI) outperforms that with Random-assigned Initialization (RI) when directly evaluating on ImageNet-1K without any tuning after initialization. For example, CI-order1 outperforms RI by $\mathbf{5.3\%}$ and $\mathbf{7.3\%}$ respectively on Des-B-9 and Des-B-10.

The effect of learngenes on Des-Nets. We train Aux-S for more epochs. As shown in Table 6, we find that better performance of Des-S-12 can be consistently achieved with better learngenes. For example, performance on CIFAR-100 can be improved from $\mathbf{87.40\%}$ to $\mathbf{88.54\%}$ with better learngenes.

5 Conclusion

In this paper, we proposed a well-motivated and highly effective Learngene approach termed SWS to initialize variable-sized Transformers, enabling adaptation to diverse resource constraints. Experimental results under various initialization settings demonstrated the effectiveness and efficiency of SWS.

Acknowledgements

This research is supported by the National Science Foundation of China (62125602, 62076063, 62206048), Natural Science Foundation of Jiangsu Province (BK20220819), Young Elite Scientists Sponsorship Program of Jiangsu Association for Science and Technology Tj-2022-027 and the Big Data Computing Center of Southeast University.

References

[Arpit *et al.*, 2019] Devansh Arpit, Víctor Campos, and Yoshua Bengio. How to initialize your network? robust initialization for weightnorm & resnets. *Advances in Neural Information Processing Systems*, 32, 2019.

[Ba *et al.*, 2016] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[Bai *et al.*, 2019] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *Advances in Neural Information Processing Systems*, 32, 2019.

[Bao *et al.*, 2022] Hangbo Bao, Li Dong, Songhao Piao, and Furu Wei. Beit: Bert pre-training of image transformers. *ICLR*, 2022.

[Bossard *et al.*, 2014] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101–mining discriminative components with random forests. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part VI 13*, pages 446–461. Springer, 2014.

[Dabre and Fujita, 2019] Raj Dabre and Atsushi Fujita. Recurrent stacking of layers for compact neural machine translation models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6292–6299, 2019.

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

[Devlin *et al.*, 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[DING *et al.*, 2023] Yahao DING, Mohammad SHIKH-BAHAEI, Zhaohui YANG, Chongwen HUANG, and Weijie YUAN. Secure federated learning over wireless communication networks with model compression. *ZTE Communications*, 21(1):46–54, 2023.

[Dosovitskiy *et al.*, 2021] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

[Feng *et al.*, 2024] Fu Feng, Jing Wang, and Xin Geng. Transferring core knowledge via learngenes. *arXiv preprint arXiv:2401.08139*, 2024.

[Glorot and Bengio, 2010] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings, 2010.

[Graham *et al.*, 2021] Benjamin Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: a vision transformer in convnet's clothing for faster inference. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12259–12269, 2021.

[He *et al.*, 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[He *et al.*, 2022] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

[Hinton *et al.*, 2015] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[Huang *et al.*, 2020] Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In *International Conference on Machine Learning*, pages 4475–4483. PMLR, 2020.

[Ji *et al.*, 2023] Zhong Ji, Jingwei Ni, Xiyao Liu, and Yanwei Pang. Teachers cooperation: team-knowledge distillation for multiple cross-domain few-shot learning. *Frontiers of Computer Science*, 17(2):172312, 2023.

[Kovaleva *et al.*, 2019] Olga Kovaleva, Alexey Romanov, Anna Rogers, and Anna Rumshisky. Revealing the dark secrets of bert. *EMNLP*, 2019.

[Krause *et al.*, 2013] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 554–561, 2013.

[Krizhevsky *et al.*, 2009] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technique Report*, 2009.

[Lan *et al.*, 2020] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *ICLR*, 2020.

[LeCun *et al.*, 2002] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 2002.

[Li *et al.*, 2024] Lei Li, Chengyu Wang, Minghui Qiu, Cen Chen, Ming Gao, and Aoying Zhou. Accelerating bert inference with gpu-efficient exit prediction. *Frontiers of Computer Science*, 18(3):183308, 2024.

[Li *et al.*, 2025] Shaoyuan Li, Yuxiang Zheng, Ye Shi, Shengjun Huang, and Songcan Chen. Kd-crowd: A knowledge distillation framework for learning from crowds. *Frontiers of Computer Science*, 19(1):191302, 2025.

[Liu *et al.*, 2021] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 10012–10022, 2021.

[Mishkin and Matas, 2015] Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

[Oquab *et al.*, 2023] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023.

[Paszke *et al.*, 2019] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33th Annual Conference on Neural Information Processing Systems*, pages 8024–8035, 2019.

[Radford *et al.*, 2021] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[Ren *et al.*, 2023] Sucheng Ren, Fangyun Wei, Zheng Zhang, and Han Hu. Tinymim: An empirical study of distilling mim pre-trained models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3687–3697, 2023.

[Samragh *et al.*, 2023] Mohammad Samragh, Mehrdad Farajtabar, Sachin Mehta, Raviteja Vemulapalli, Fartash Faghri, Devang Naik, Oncel Tuzel, and Mohammad Rastegari. Weight subcloning: direct initialization of transformers using larger pretrained ones. *arXiv preprint arXiv:2312.09299*, 2023.

[Shi *et al.*, 2024] Boyu Shi, Shiyu Xia, Xu Yang, Haokun Chen, Zhiqiang Kou, and Xin Geng. Building variable-sized models via learngene pool. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 14946–14954, 2024.

[Takase and Kiyono, 2021] Sho Takase and Shun Kiyono. Lessons on parameter sharing across layers in transformers. *arXiv preprint arXiv:2104.06022*, 2021.

[Touvron *et al.*, 2021] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *International conference on machine learning*, pages 10347–10357. PMLR, 2021.

[Touvron *et al.*, 2023] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[Wang *et al.*, 2022a] Qiu-Feng Wang, Xin Geng, Shu-Xia Lin, Shi-Yu Xia, Lei Qi, and Ning Xu. Learngene: From open-world to your learning task. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8557–8565, 2022.

[Wang *et al.*, 2022b] Wenhui Wang, Hangbo Bao, Li Dong, Johan Bjorck, Zhiliang Peng, Qiang Liu, Kriti Aggarwal, Owais Khan Mohammed, Saksham Singhal, Subhojit Som, et al. Image as a foreign language: Beit pretraining for all vision and vision-language tasks. *arXiv preprint arXiv:2208.10442*, 2022.

[Wang *et al.*, 2023] Qiufeng Wang, Xu Yang, Shuxia Lin, and Xin Geng. Learngene: Inheriting condensed knowledge from the ancestry model to descendant models. *arXiv preprint arXiv:2305.02279*, 2023.

[Xia *et al.*, 2024] Shiyu Xia, Miaosen Zhang, Xu Yang, Ruiming Chen, Haokun Chen, and Xin Geng. Transformer as linear expansion of learngene. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 16014–16022, 2024.

[Xu *et al.*, 2023] Zhiqiu Xu, Yanjie Chen, Kirill Vishniakov, Yida Yin, Zhiqiang Shen, Trevor Darrell, Lingjie Liu, and Zhuang Liu. Initializing models with larger ones. *arXiv preprint arXiv:2311.18823*, 2023.

[Yan *et al.*, 2023] Jintao Yan, Tan Chen, Bowen Xie, Yuxuan Sun, Sheng Zhou, and Zhisheng Niu. Hierarchical federated learning: Architecture, challenges, and its implementation in vehicular networks. *ZTE Communications*, 21(1):38–45, 2023.

[Zhang *et al.*, 2022] Jinnian Zhang, Houwen Peng, Kan Wu, Mengchen Liu, Bin Xiao, Jianlong Fu, and Lu Yuan. Minivit: Compressing vision transformers with weight multiplexing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12145–12154, 2022.