Slice-and-Pack: Tailoring Deep Models for Customized Requirements

Ruice Rao[*1,2], **Dingwei Li**[*1,2], **Ming Li**[†1,2]

[1]National Key Laboratory for Novel Software Technology, Nanjing University, China
[2]School of Artificial Intelligence, Nanjing University, China
{raorc, lidw, lim}@lamda.nju.edu.cn

Abstract

The learnware paradigm aims to establish a learnware market such that users can build their own models by reusing appropriate existing models in the market without starting from scratch. It is often the case that a single model is insufficient to fully satisfy the user's requirement. Meanwhile, offering multiple models can lead to higher costs for users alongside an increase in hardware resource demands. To address this challenge, this paper proposes the "Slice-and-Pack" (S&P) framework to empower the market to provide users with only the required model fragments without having to offer entire abilities of all involved models. Our framework first slices a set of models into small fragments and subsequently packs selected fragments according to user's specific requirement. In the slicing stage, we extract units layer by layer and connect these units to create numerous fragments. In the packing stage, an encoder-decoder mechanism is employed to assemble these fragments. These processes are conducted within data-limited constraints due to privacy concerns. Extensive experiments validate the effectiveness of our framework.

Introduction

Machine learning has achieved great success in various domains (Bengio, Lecun, and Hinton 2021; Silver et al. 2016; OpenAI 2023; Luo et al. 2022; Radford et al. 2021). However, building a satisfactory learning model from scratch is often time-consuming and requires significant expertise, and personalized customization is often necessary for specific situations. Moreover, the access to raw data is often limited by privacy concerns, making it even challenging to train well-behaved models from scratch. To address these issues, Zhou (2016) proposed the *learnware* paradigm, aiming to establish a learnware market such that users can build their own models by reusing appropriate existing models in the market without starting from scratch. Unlike LLM, the learnware market enables model construction and reuse in broader application scenarios such as physics, chemistry, geology, medicine, manufacturing, and more. A learnware is a pre-trained model associated with a specification describing the model's specialty and utility. Developers can submit trained models to the market with a few data points describing their tasks, and the market will assign a specification upon accepting this submitted model. When users try to tackle a learning task, the market can recommend helpful learnwares whose specifications match the requirement.

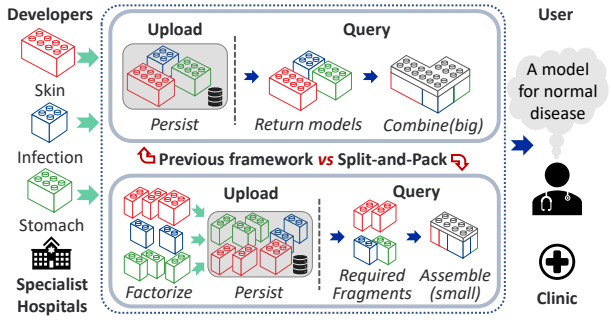

Figure 1: Comparison of the two frameworks. S&P provides more compact models with only the necessary abilities, resulting in cost savings and lower device requirements.

However, in many cases it is hard to find a single model that fully meets the user's requirements, especially in the early stages of the learnware market. In real-world scenarios, the abilities desired by users might spread across several models. Offering all involved models would make users feel compelled to purchase for potentially redundant abilities, and also requires more hardware resource for deployment. For instance, consider the community clinics that try to use patient records for diagnosis in Figure 1. Due to the constraints in equipment and treatment capabilities, these clinics are primarily tasked with addressing *normal diseases*. In the learnware market, there are several existing well-behaved models available from specialist hospitals trained on private data. However, these models are designed to diagnose *as many diseases as possible* in a specific field. Since the requirements of community clinics are abilities to diagnose normal diseases, these rare abilities of the model are redundant for them. It is to be mentioned that models equipped with more capabilities typically demand more computational power for inference, leading to the need for more robust devices and higher acquisition costs. As highlighted in (Liang et al. 2022), the cost of commercial models significantly escalates as the size and capability of the mod-

[*]These authors contributed equally.
[†]Ming Li is the corresponding author.

Setting	Data limited	Increasing diversity	Cheap	Few parameters	Plug&Play	Heterogeneous model
Train from scratch	✗	-	✗	-	✗	-
Transfer learning	✓	✗	✓	✓	✗	✗
Model reuse	✓	✗	✗	✓	✗	✓
Model decomposition	✓	✗	✓	✓	✗	✗
Ensemble	✓	✓	✗	✗	✓	✓
S&P	✓	✓	✓	✓	✓	✓

Table 1: Characteristics of problem settings

els increase, ranging from $169 to upwards of $10,000. Furthermore, the utilization of these stronger models often necessitates much more computational resources, which could impose further financial burden on community clinics.

To address this issue, we propose the "Slice-and-Pack" (S&P) framework, which empowers the market to provide users with only the required model fragments, eliminating the need to offer entire abilities of all involved models. These fragments can be combined in various ways to satisfy different users' specific requirements. As illustrated in Figure 1, to meet the clinic's requirements, we can slice models into fragments of different diseases and combine fragments of normal diseases. This framework offers superior flexibility and convenience compared to using the entire model, minimizing the cost of redundant abilities while providing easy plug-and-play functionality. Whenever the change of requirements happens, users can easily discard outdated fragments and purchase new ones as needed. Furthermore, the framework can also greatly benefit the market development, as there is no need for the market to consider all possible combinations of abilities, thereby expanding the market's capacity to match that of a market several times its size.

To achieve above desired properties, our framework works in two stages: slicing and packing. In the slicing stage, we extract fragments from the original model, each corresponding to a sub-abilities. We first identify units that are significant to the sub-abilities layer by layer and then progressively connect each layer's selected units. This process creates individual fragments that perform specific abilities. In the packing stage, we use an encoder-decoder mechanism to pack the previously stored fragments. We take the fragments identified in the slicing part and assemble them to form a new model tailored to the user's specific requirements. All these processes simply use the limited data provided by developers for constructing the learnware specification. We summarize our contributions in the following.

- We are the first to tackle the problem of increasing model diversity in the market, which holds significant commercial viability compared to previous problems.
- We introduce the "Slice-and-Pack" (S&P) framework, a novel method with plug-and-play feature that expands the market in data-limited environments.
- Our empirical evaluations show that the S&P can generate highly accurate packed models and expand the market's capacity by many times. Moreover, the minimal time required for packing makes it readily available to users.

Problem Setting

The learnware paradigm (Zhou 2016; Zhou and Tan 2022) presents a promising framework where a vast number of models are submitted by developers from various tasks without the availability of their original training data. It poses significant challenges for users to identify and reuse helpful models in the market (Wu et al. 2021; Tan et al. 2022). Assume that there are N models $\{\mathcal{M}^{(i)}\}_{i=1}^{N}$ developed by different developers in the learnware market. Each model $\mathcal{M}^{(i)}$ has an ability set representing the abilities that the model has. In the following, we focus on classification task and each ability is the function to identify a single class. Let $\mathcal{C}^{(i)}$ be the class set of i-th model $\mathcal{M}^{(i)}$, $\mathcal{U} = \bigcup_{i=1}^{N} \mathcal{C}^{(i)}$ be the set of all classes in $\{\mathcal{M}^{(i)}\}_{i=1}^{N}$, $\mathcal{D}^{(i)}$ be the limited samples of model $\mathcal{M}^{(i)}$ uploaded by developers, and $\mathcal{D}_c^{(i)} \subset \mathcal{D}^{(i)}$ be the samples of class $c \in \mathcal{C}^{(i)}$.

The problem in this paper is based on the learnware paradigm: Given a set of models and a few samples $\{(\mathcal{M}^{(i)}, \mathcal{D}^{(i)})\}_{i=1}^{N}$, how to get a set of fragments, allowing for the rapid and easy construction of a smallest possible model with any subset of classes $\mathcal{C} \subset \mathcal{U}$.

Let $\mathcal{F}^c$ be the fragment of class c. Our goal can be formalized as optimizing the following objective:

$$\min_{\mathcal{F}^c} \mathbb{E}_{\mathcal{C}\subset\mathcal{U},(x,y)\sim\mathcal{P}_\mathcal{C}} \mathcal{L}\left(\mathcal{F}_\mathcal{C}(x), y\right) \tag{1}$$

where $\mathcal{P}_\mathcal{C}$ is the distribution of class c's samples, $\mathcal{F}_\mathcal{C} = \text{Union}(\{\mathcal{F}^c\}_{c\in\mathcal{C}})$ is the final model by packing required fragments, $\mathcal{L}(\cdot,\cdot)$ is some loss function.

Related Work

The difference between other settings is shown in Table 1. Transfer learning (Pan and Yang 2010) and domain adaptation (Ben-David et al. 2006; Sun et al. 2023) are techniques aiming to transfer knowledge from the source domain to the target domain. They assume that raw data from one (Huang et al. 2006; Pan et al. 2010; Fernando et al. 2013) or multiple domains (Shu et al. 2021; Nguyen et al. 2021; Yang et al. 2022) are accessible when training the target model. However, in the learnware paradigm, the raw source data is unavailable at the time of training the target model, making these techniques inapplicable.

Hypothesis transfer learning (Kuzborskij and Orabona 2013, 2017) and model reuse (Ding and Zhou 2020; Zhao, Cai, and Zhou 2020) attempt to exploit pre-trained models

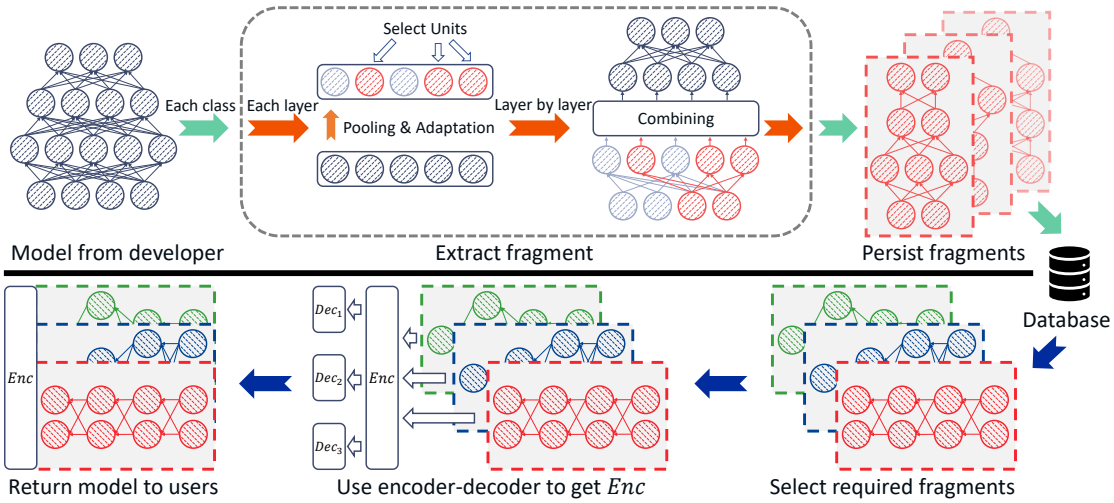

Figure 2: The overview of Slice-and-Pack.

to handle current jobs of learners. These techniques assume that the pre-trained models are helpful for the current job (Li et al. 2020a; Shen et al. 2021). However, this assumption is not suitable for our problem as the abilities required by users may be distributed across several models, making it difficult to exactly find a single model that is helpful to users. Additionally, we aim to slice pre-trained models into smaller models with different sub-abilities, rather than replicating the original functionality.

Multi-party learning (Pathak, Rane, and Raj 2010; Wu, Liu, and Zhou 2019) aims to unite local data to solve the same or similar task in privacy-preserving ways, rather than using the existing pre-trained models. Some recent attempts show some possibilities of leveraging extracted components of existing models for specific tasks (Yang, Ye, and Wang 2022). These approaches necessitate developers to share substantial amounts of data, which is unacceptable for learnware markets due to privacy concerns of the model provider.

Model decomposition is a technique used to increase parameter-efficiency in deep learning models by breaking down parameter-heavy layers into multiple lightweight ones. This approach involves using various techniques such as low-rank decomposition (Li et al. 2020b) and weight decomposition (Kim et al. 2015; Zhang et al. 2015; Hameed et al. 2022). They aim to generate smaller models of similar abilities with comparable performance for distributed runtime environments, but not models with different functionalities.

The Slice-and-Pack Framework

In this section, we present the details of our Slice-and-Pack (S&P) framework, which is composed of two parts: Slicing and Packing, as illustrated in Figure 2. The algorithm is shown in Appendix A.

Slicing the Network

In this subsection, we denote $\mathcal{M}$, $\mathcal{D}$ as the shorthand for $\mathcal{M}^{(i)}$, $\mathcal{D}^{(i)}$. The slicing method aims to obtain the fragment $\mathcal{F}^c$ with class c from the network model $\mathcal{M}$. In this paper, fragment refers to the model obtained from the feature extraction component of the original network. We regard $\mathcal{M}$ as a series of layers m_i composed of a linear layer and some nonlinear layers. Almost all network layers can be seen as a collection of units. The unit is a filter for the convolution layer, and the unit is a neuron for the linear layer. Suppose $\mathcal{M}$ has L layers, we can express $\mathcal{M}$ as:

$$\mathcal{M} = m_1 \circ m_2 \circ \cdots \circ m_{L-1} \circ m_L$$

We adopt a dual-stage strategy to extract the fragment $\mathcal{F}^c$ of class c from the network model $\mathcal{M}$, which is designed to ease the risk of overfitting. In the first stage, we select units which are important to the class c layer by layer. For each layer m_i, we use techniques *Ability Pooling* and *Ability Adaption* to get selected units layer f_i, as shown in Figure 3. In the second stage, each f_i is progressively combined and finally we get the fragment $\mathcal{F}^c$ of class c.

Ability Pooling. In Slice-and-Pack setting, it is necessary to identify the units that are important to a specific class c rather than the whole original model $\mathcal{M}$. While some methods use parameter weights as regularization or criticism to make the model sparser or clip it, high-value parameters are essential to the overall model rather than a specific class. To address this, we use the activation of class c's samples at the i-th layer m_i as the means of identifying important units. We first use this activation as a regularization, and then we select units based on it.

Let $\mathcal{Z}_i^c$ represent the output of layer m_i with $\mathcal{D}_c$ as input, $\mathcal{M}'$ be the model having parameters completely the same as $\mathcal{M}$, and u_i be the number of units in m_i. To enhance the sparsity of weights among each units rather than the whole layer, we use $\ell_{2,1}$-norm (Yang et al. 2011) of $\mathcal{Z}_i^c$ as follows:

$$\|\mathcal{Z}_i^c\|_{2,1} = \frac{1}{|\mathcal{Z}_i^c|} \sum_{\mathbf{z} \in \mathcal{Z}_i^c} \sum_{j=1}^{u_i} \|\mathbf{z}^j\|_2 , \tag{2}$$

where $\mathbf{z}^j$ is the output of the j-th unit. We fine-tune m_i with the pooling loss containing $\ell_{2,1}$-norm of $\mathcal{Z}_i^c$ and the ℓ_2-loss

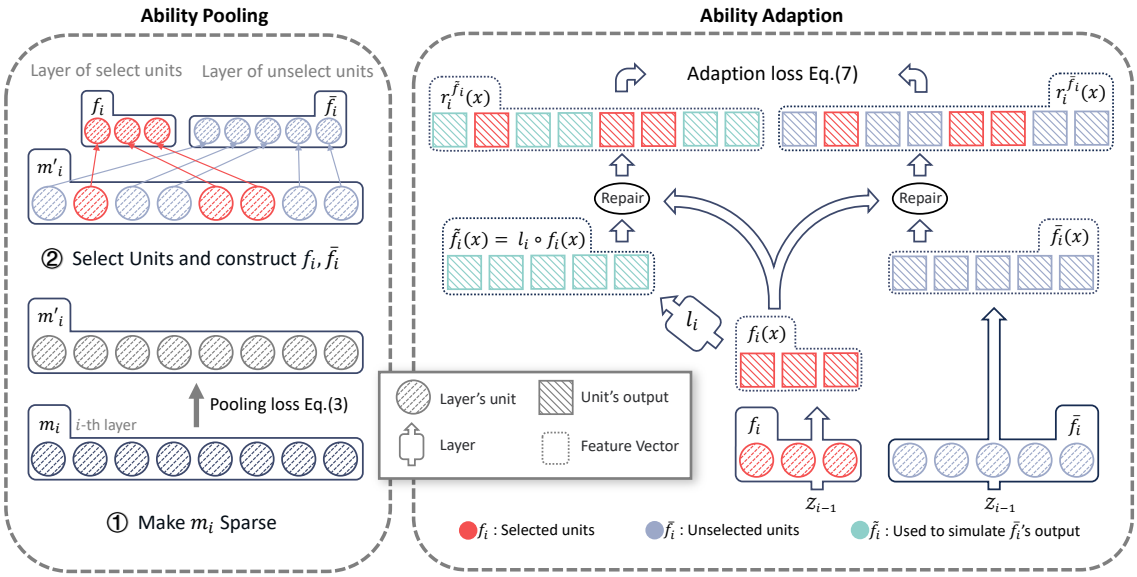

Figure 3: The process of factorizing layer m_i. In ability pooling, we will centralize the functionality of class c to less units and then select units to get f_i. In ability adaption, We use the adaption loss to adapt f_i for the change of structure.

function $\mathcal{L}_i^P$ between $\mathcal{M}$ and $\mathcal{M}'$, expressed as

$$\mathcal{L}_i^P = \mathbb{E}_{x\in\mathcal{D}}\left[\|\mathcal{M}'(x) - \mathcal{M}(x)\|_2^2\right] + \alpha\,\|\mathcal{Z}_i^c\|_{2,1}\,, \quad (3)$$

where α is a hyperparameter. We can obtain a sparser layer m_i' from $\mathcal{M}'$ by using the loss function $\mathcal{L}_i^P$.

Then we select units by either threshold or rate. Let U_i be the set of selected unit indices as follows:

$$U_i = \left\{j \mid |(\mathcal{Z}_i^c)_j| > 1-\beta\right\} \quad \text{(threshold)} \quad (4)$$

$$U_i = \left\{j \mid j \in \text{Top-}\beta\left(\{|(\mathcal{Z}_i^c)_j|\}_{j=1}^{u_i}\right)\right\}, \quad \text{(rate)} \quad (5)$$

where β is a hyperparameter in the range from 0 to 1. Finally we get the new layer f_i from m_i' for class c.

Ability Adaption. After selecting the units, we need to adapt the layer f_i to account for changes in the structure of the neural network. However, there are two challenges that we need to address: *size mismatch* and *ability collapse*.

Size mismatch refers to the discrepancy between the output size of the i-th layer f_i and the input size of the $(i+1)$-th layer f_{i+1}. Simply removing the parameters of $(i+1)$-th layer without considering the context of the layer's situation is wasteful, especially when dealing with limited data. To solve this problem, we assume that selected units contain sufficient information. This allows us to recover the output of dropped units by using output of the remaining units. We achieve this by employing a linear layer l_i. It takes the output of f_i as input and produces features with the same size as the dropped units. The output of l_i can then be used to fill in the missing results from the removed units in m_i', which we refer to as the repair operation, as shown in the right side of Figure 3. Moreover, l_i can be fused into the next layer, thereby avoiding the introduction of additional parameters (Li et al. 2020a). The proof is in Appendix B. Let $\widetilde{f}_i = l_i \circ f_i$, and we use $r_i^{\widetilde{f}_i}$ to refer to the repair of f_i with $\widetilde{f}_i$.

Ability collapse refers to the situation that the ability of the extracted layer f_i deteriorates, especially when available data is limited. Previous approaches mitigate this issue by ensuring that the predictions of new layers or models remain consistent with original models (Hinton, Vinyals, and Dean 2015; Zhang, Zhu, and Ye 2019). However, in our situation, directly aligning f_i and m_i as their semantic spaces is not appropriate, due to the fact that the extracted layer f_i focuses on a different task compared to m_i. To solve this problem, we assume that different classes share some common abilities. This is particularly evident in the shallow layers of the network, where more low-level features are being abstracted. For example, in a DNN classifier for cars, horses, and dogs, information about edges and corners is captured first, followed by information about colors and contours, and finally information about object parts. Therefore, it is highly probable that these three classes share common sub-abilities to extract features, such as the ability to capture contours. We can put f_i back into the model and fine-tune it using samples from other classes to address this issue. By doing so, f_i will be updated when there is a sub-ability in f_i that is also a sub-ability of another class. Let $\overline{f}_i$ be the layer composed of the dropped units from m_i during the Ability pooling process. We simultaneously use $\overline{f}_i$ to repair f_i, denoted as $r_i^{\overline{f}_i}$.

Let $\mathcal{M}(\,\cdot\,; p_{i:j})$ represent the replacement of layers from i-th layer to j-th layer with repair $r_i^{p_i}, \cdots, r_j^{p_j}$, which can be expressed as

$$\mathcal{M}(\,\cdot\,; p_{i:j}) = m_1 \circ \cdots \circ m_{i-1} \circ r_i^{p_i} \circ \cdots \circ r_j^{p_j} \circ m_{j+1} \circ \cdots \circ m_L, \quad (6)$$

so we can utilize $\mathcal{M}(\,\cdot\,; \widetilde{f}_{i:j})$ to represent a model that employs $\widetilde{f}_i$ defined in size mismatch to repair layers, and use $\mathcal{M}(\,\cdot\,; \overline{f}_{i:j})$ to denote a model that utilizes $\overline{f}_i$ to repair lay-

ers. Now we can define the adaption loss $\mathcal{L}_i^A$ as:

$$\mathcal{L}_i^A = \mathbb{E}_{(x,y)\in\mathcal{D}}\left[\mathcal{L}_{\text{cls}}(x,y;i,i) + \gamma\mathcal{L}_{\text{dis}}(x;i,i)\right] \quad (7)$$

$$\mathcal{L}_{\text{cls}}(x,y;i,j) = \mathcal{L}_c\left(\mathcal{M}(x;\widetilde{f}_{i:j})\circ\mathcal{H}, y\right) \quad (8)$$

$$\mathcal{L}_{\text{dis}}(x;i,j) = \left\|\mathcal{M}(x;\overline{f}_{i:j}) - \mathcal{M}(x)\right\|_2^2, \quad (9)$$

where $\mathcal{L}_c$ is the cross-entropy loss, γ is the hyperparameter, and $\mathcal{H}$ is a classifier to output classification result. $\mathcal{L}_{\text{cls}}$ works as the classification error to train ℓ_i and f_i. $\mathcal{L}_{\text{dis}}$ works as the reconstruction loss to update f_i by utilizing common subabilities from dropped units, solving ability collapse problem. All other classes are considered as a single negative class.

Combining f_i layer by layer. A series of extracted layer f_i has been obtained, we start to construct $\mathcal{F}^c$. However, since each f_i is trained separately, we need to combine them one by one from front to end. The combining loss function $\mathcal{L}_i^C$ can be expressed as

$$\mathcal{L}_i^C = \mathbb{E}_{(x,y)\in\mathcal{D}}\left[\mathcal{L}_{\text{cls}}(x,y;1,i) + \gamma\mathcal{L}_{\text{dis}}(x;1,i)\right]. \quad (10)$$

By optimizing the loss function from $\mathcal{L}_1^C$ to $\mathcal{L}_L^C$, we get optimized model $\mathcal{M}(\,\cdot\,;p_{1:L})$. Since l_i is the linear layer and repair operation is also the linear operation, we can fuse all l_i into the linear layer next to it and finally get $\mathcal{F}^c$.

Packing Fragments

The packing method aims to construct a new model by fragments $\{\mathcal{F}^c\}_{c\in\mathcal{C}}$ so that the new model has all abilities in $\mathcal{C}$. The challenge here is how to effectively combine features generated by these fragments. One possible approach is to simply concatenate features, but it will result in a very long feature vector that is difficult to work with, especially when there are multiple fragments.

In order to obtain tight features, we minimize the reconstruction loss for each fragment. Let Enc be the fusion layer used to combine the features, and let $\{Dec_c\}_{c\in\mathcal{C}}$ be the decoders used to restore the output of each fragment $\mathcal{F}^c$. The final model $\mathcal{F}_{\mathcal{C}}$, use $\mathcal{F}$ for simplicity, can be expressed as:

$$\mathcal{F}(x) = Enc\left(\text{concat}\left(\{\mathcal{F}^c(x)\}_{c\in\mathcal{C}}\right)\right). \quad (11)$$

To ensure that the output of $\mathcal{F}$ contains all the information from the $\mathcal{F}^c$ fragments, we minimize the distance between $\mathcal{F}^c$ and $\mathcal{F}\circ Dec_c$. We define the packing loss function as:

$$\mathcal{L}^K = \mathbb{E}_{c\in\mathcal{C},(x,y)\in\mathcal{D}_c}\left[\mathcal{L}_c^K(x,y)\right] \quad (12)$$

$$\mathcal{L}_c^K(x,y) = \mathcal{L}_c\left(\mathcal{F}(x)\circ\mathcal{H}^K, y\right) + \delta\left\|\mathcal{F}(x)\circ Dec_c - \mathcal{F}^c\right\|_2^2$$

in which $\mathcal{H}^K$ is the classifier for the packed model, and δ is the hyperparameter.

In the process, we only update the parameters of Enc, $\{Dec_c\}_{c\in\mathcal{C}}$ and $\mathcal{H}^K$, while leaving the fragments $\{\mathcal{F}^c\}_{c\in\mathcal{C}}$ unchanged. This allows for easy integration or removal of fragments to meet new requirements, promoting high reusability of these fragments. We also have the flexibility to combine fragments with different structures. Additionally, due to limited number of parameters in Enc, $\{Dec_c\}_{c\in\mathcal{C}}$ and $\mathcal{H}^K$, with each being a linear layer followed by an activation layer, the workload required to adjust parameters is significantly low, resulting in increased efficiency in practice.

Experiments

We conducted extensive experiments on various datasets for image classification and sentiment analysis. Our code was implemented by PyTorch and executed on an NVIDIA A100 40GB PCIe GPU with AMD EPYC 7H12 64-Core Processor. The implement details are provided in Appendix C.

Experimental Settings

Datasets, Model and Task. We conducted a series of experiments on four different datasets: CIFAR10, CIFAR100 (Krizhevsky, Hinton et al. 2009), TREC (Hovy et al. 2001), and SST-5 (Socher et al. 2013). CIFAR10 and CIFAR100 consist of 50,000 images for training and 10,000 for testing. The TREC Question Classification dataset contains 5,500 sentences in the training set and another 500 in the test set, with 6 classes. SST-5 dataset consists of 8,544 sentences in the training set and another 2,210 in the test set, with 5 classes. We use VGG16 (Simonyan and Zisserman 2014) and ResNet34 (He et al. 2016) as original models on CIFAR-10 and CIFAR-100, and CNN (Kim 2014) and RNN as original models on TREC and SST-5.

We have designed 7 tasks to construct target models from original models. Task T_1 uses 3 original models trained on CIFAR10. Tasks T_2 and T_3 utilize 5 original models trained on CIFAR100, with the former having models from the same superclass and the latter having models from different superclasses. Task T_4 involves 10 original models trained on CIFAR100. Tasks T_6 and T_7 use 2 original models trained on TREC and SST-5, respectively. In T_7, the two models share a common class. Our approach involves slicing the models into multiple fragments, with each fragment representing a class, and then packing the selected fragments. This is in contrast to other methods that directly obtain the target model without the slicing and packing process. For each original model, we randomly sample k samples per class from corresponding dataset. More details and experiments with different datasets and architectures are in Appendix D.

Experimental Results

Performance on Image data. We begin by presenting the results of our method on image data and compare it with the finetuning method, CA-MKD (Zhang, Chen, and Wang 2022), and NetGraft (Shen et al. 2021). Finetuning method fine-tunes one of the original models using data of the target task. CA-MKD is the multi-teacher knowledge distillation method which adaptively assigns sample-wise reliability for each teacher prediction with the help of ground-truth labels. NetGraft is an knowledge distillation approach utilizing limited data. In our experiments, we utilize NetGraft to partition models and train a classifier layer for the combining models. As shown in Table 2, our method achieves better accuracy with a small number of parameters, especially for S&P (T). While S&P (R) has relatively lower accuracy, its parameter count is smaller. It is worth noting that the standard deviation of many results is higher than usual because we calculate the value between different models in the task, rather than different seeds in typical experimental settings. CA-MKD requires a substantial amount of data to

Task	Method	VGG16 Param ($\times 10^6$)	VGG16 Acc.(%) $k=5$	VGG16 Acc.(%) $k=10$	VGG16 Acc.(%) $k=20$	ResNet34 Param ($\times 10^6$)	ResNet34 Acc.(%) $k=5$	ResNet34 Acc.(%) $k=10$	ResNet34 Acc.(%) $k=20$
T_1	CA-MKD	$14.7_{(33.3\%)}$	(failed)	(failed)	(failed)	$18.9_{(33.3\%)}$	(failed)	(failed)	(failed)
	NetGraft	$11.5_{(26.1\%)}$	$47.21_{\pm 11.1}$	$49.46_{\pm 6.73}$	$52.18_{\pm 11.6}$	-	-	-	-
	Finetune	$14.7_{(33.3\%)}$	$65.47_{\pm 6.41}$	$66.88_{\pm 4.47}$	$66.99_{\pm 2.33}$	$18.9_{(33.3\%)}$	$67.29_{\pm 4.69}$	$70.04_{\pm 6.74}$	$71.56_{\pm 4.80}$
	S&P (R)	$11.5_{(26.1\%)}$	$\mathbf{71.13}_{\pm 2.55}$	$\mathbf{74.90}_{\pm 6.56}$	$74.47_{\pm 7.52}$	$19.0_{(33.5\%)}$	$\mathbf{75.43}_{\pm 4.91}$	$\mathbf{78.49}_{\pm 8.15}$	$\mathbf{80.43}_{\pm 6.01}$
	S&P (T)	$7.4_{(16.8\%)}$	$62.20_{\pm 4.00}$	$73.23_{\pm 5.38}$	$\mathbf{76.37}_{\pm 5.49}$	$25.7_{(45.3\%)}$	$73.57_{\pm 2.25}$	$76.89_{\pm 6.39}$	$80.31_{\pm 6.43}$
T_2	CA-MKD	$14.7_{(20.0\%)}$	(failed)	(failed)	$63.10_{\pm 10.2}$	$18.9_{(20.0\%)}$	(failed)	(failed)	$53.35_{\pm 10.5}$
	NetGraft	$19.2_{(26.1\%)}$	$54.45_{\pm 7.94}$	$62.30_{\pm 10.1}$	$63.90_{\pm 6.75}$	-	-	-	-
	Finetune	$14.7_{(20.0\%)}$	$61.10_{\pm 8.02}$	$64.00_{\pm 9.52}$	$68.35_{\pm 8.21}$	$18.9_{(20.0\%)}$	$65.20_{\pm 9.69}$	$71.40_{\pm 8.64}$	$73.30_{\pm 9.60}$
	S&P (R)	$19.2_{(26.1\%)}$	$61.95_{\pm 12.9}$	$68.65_{\pm 11.0}$	$72.90_{\pm 7.79}$	$31.7_{(33.5\%)}$	$67.10_{\pm 13.7}$	$72.55_{\pm 9.26}$	$75.85_{\pm 9.92}$
	S&P (T)	$27.4_{(37.2\%)}$	$\mathbf{68.40}_{\pm 12.0}$	$\mathbf{78.10}_{\pm 6.08}$	$\mathbf{81.40}_{\pm 4.15}$	$58.6_{(62.0\%)}$	$\mathbf{77.00}_{\pm 10.6}$	$\mathbf{78.90}_{\pm 10.2}$	$\mathbf{84.50}_{\pm 7.92}$
T_3	CA-MKD	$14.7_{(20.0\%)}$	(failed)	(failed)	$62.42_{\pm 7.37}$	$18.9_{(20.0\%)}$	(failed)	(failed)	$50.41_{\pm 10.5}$
	NetGraft	$19.2_{(26.1\%)}$	$49.50_{\pm 5.16}$	$56.85_{\pm 10.3}$	$61.85_{\pm 5.59}$	-	-	-	-
	Finetune	$14.7_{(20.0\%)}$	$60.55_{\pm 4.26}$	$62.05_{\pm 3.62}$	$67.00_{\pm 5.31}$	$18.9_{(20.0\%)}$	$64.55_{\pm 7.17}$	$68.15_{\pm 5.59}$	$73.55_{\pm 6.44}$
	S&P (R)	$19.2_{(26.1\%)}$	$64.80_{\pm 8.32}$	$70.05_{\pm 8.33}$	$72.15_{\pm 6.70}$	$31.7_{(33.5\%)}$	$72.80_{\pm 5.66}$	$72.20_{\pm 5.38}$	$79.40_{\pm 5.26}$
	S&P (T)	$27.0_{(36.7\%)}$	$\mathbf{71.35}_{\pm 6.43}$	$\mathbf{77.80}_{\pm 7.41}$	$\mathbf{79.55}_{\pm 4.27}$	$59.8_{(63.3\%)}$	$\mathbf{75.75}_{\pm 8.01}$	$\mathbf{81.20}_{\pm 4.45}$	$\mathbf{84.65}_{\pm 3.92}$
T_4	CA-MKD	$14.7_{(20.0\%)}$	(failed)	(failed)	$59.08_{\pm 8.68}$	$18.9_{(20.0\%)}$	(failed)	(failed)	$47.74_{\pm 11.3}$
	NetGraft	$19.2_{(26.1\%)}$	$44.40_{\pm 6.01}$	$55.36_{\pm 7.37}$	$58.84_{\pm 6.57}$	-	-	-	-
	Finetune	$14.7_{(20.0\%)}$	$50.34_{\pm 5.93}$	$54.36_{\pm 5.42}$	$58.72_{\pm 5.39}$	$18.9_{(20.0\%)}$	$53.90_{\pm 5.63}$	$59.04_{\pm 4.42}$	$63.32_{\pm 5.47}$
	S&P (R)	$19.2_{(26.1\%)}$	$56.40_{\pm 7.23}$	$63.74_{\pm 7.96}$	$69.86_{\pm 7.03}$	$31.7_{(33.5\%)}$	$66.80_{\pm 9.56}$	$69.15_{\pm 7.52}$	$77.30_{\pm 6.17}$
	S&P (T)	$14.8_{(20.1\%)}$	$\mathbf{60.62}_{\pm 5.98}$	$\mathbf{65.96}_{\pm 6.47}$	$\mathbf{73.72}_{\pm 5.38}$	$53.1_{(56.2\%)}$	$\mathbf{70.10}_{\pm 6.15}$	$\mathbf{74.90}_{\pm 5.73}$	$\mathbf{79.46}_{\pm 4.12}$
T_5	CA-MKD	$14.7_{(10.0\%)}$	(failed)	$42.22_{\pm 5.19}$	$50.26_{\pm 4.83}$	$18.9_{(10.0\%)}$	(failed)	$34.01_{\pm 5.88}$	$40.97_{\pm 6.11}$
	NetGraft	$38.4_{(26.1\%)}$	$43.42_{\pm 6.65}$	$53.18_{\pm 8.95}$	$59.16_{\pm 6.69}$	-	-	-	-
	Finetune	$14.7_{(10.0\%)}$	$34.54_{\pm 4.20}$	$37.93_{\pm 4.41}$	$42.11_{\pm 3.44}$	$18.9_{(10.0\%)}$	$40.04_{\pm 4.95}$	$45.38_{\pm 5.07}$	$50.81_{\pm 4.44}$
	S&P (R)	$38.4_{(26.1\%)}$	$45.59_{\pm 5.83}$	$54.63_{\pm 6.27}$	$59.22_{\pm 5.41}$	$63.4_{(33.5\%)}$	$50.67_{\pm 4.99}$	$57.96_{\pm 5.80}$	$63.40_{\pm 4.68}$
	S&P (T)	$27.7_{(18.8\%)}$	$\mathbf{48.31}_{\pm 5.58}$	$\mathbf{56.35}_{\pm 5.32}$	$\mathbf{65.03}_{\pm 4.75}$	$101.8_{(53.9\%)}$	$\mathbf{59.67}_{\pm 6.03}$	$\mathbf{66.10}_{\pm 4.89}$	$\mathbf{71.73}_{\pm 4.25}$

Table 2: The performance on image datasets. S&P (R) denotes selecting units with rate, and S&P (T) denotes selecting units with threshold. The percentage value in parentheses of the parameter column indicates the number of parameters compared to the ensemble method. (failed) means that the method cannot run in the setting for the lack of sufficient samples. Due to the code for ResNet34 is not provided, there are no reported results using NetGraft on this architecture.

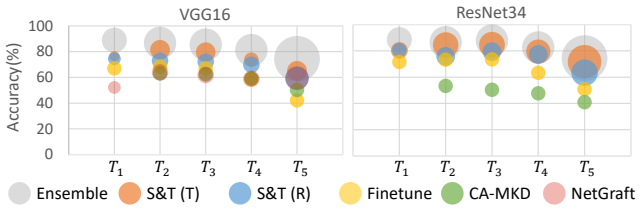

Figure 4: Comparison of parameter amount and accuracy.

Task	Method	Param ($\times 10^4$)	Acc.(%) $k=5$	Acc.(%) $k=10$
T_6 (RNN)	Ensemble	16.0	80.38	83.47
	Finetune	8.00	64.53	69.14
	S&P	6.95	79.79	81.56
T_7 (CNN)	Ensemble	72.0	69.11	73.32
	Finetune	36.0	50.49	52.21
	S&P	35.3	67.73	74.02

Table 3: Result on text datasets.

obtain the corresponding models. It cannot get result when the number of samples is small. NetGraft is designed to obtain smaller models while maintaining the same ability as original models. We also compare our results with the ensemble method when $k = 20$, which integrates results of all original models, as shown in Figure 4. Although the ensemble method achieves nearly 5 points higher accuracy than S&P, the resulting model is significantly larger, with 2 to 10 times more parameters than other generated methods. Furthermore, the ensemble method means users must purchase all models, resulting in higher costs. Experiment of packing fragments with different architectures or from different datasets is shown in Appendix D.

Performance on Textual data. We evaluate the performance of our approach on text datasets TREC and SST-5 datasets for Task T_6 and T_7. Table 3 presents the accuracy of our method in comparison with the ensemble and finetune methods. Ensemble method concatenates the features of the original models and trains a classifier based on the concatenated feature, while finetune method fine-tunes one of the original models. Our method achieves comparable results with the ensemble method and even outperforms the ensemble method on T_7 when $k = 10$. The results of finetune method show that the provided samples are insufficient to train a new model.

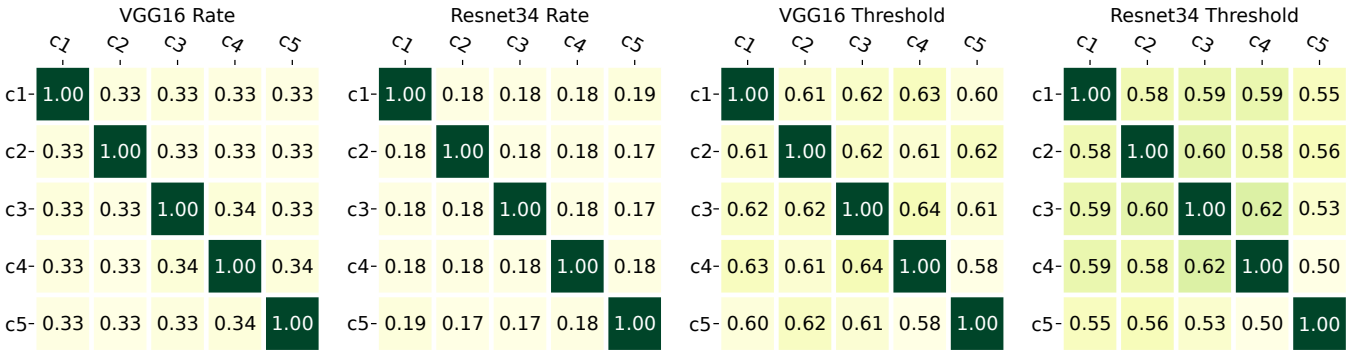

Figure 5: The Intersection over Union (IoU) of the selected units set U_i between fragments varies significantly based on the fragment's class. The IoU of S&P with threshold is relatively high, as it maintains more parameters in the front layers.

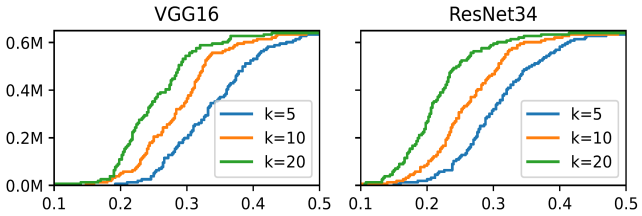

Figure 6: Equivalent expansion rate (Y-axis) of the market on different market Acceptable Error (X-axis).

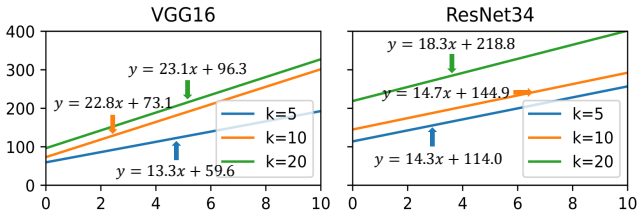

Figure 7: The spent time per class (Y-axis, seconds) as the number of user required models (X-axis) increases on T_1.

The Equivalent expansion rate of the Market. The equivalent expansion rate is a measure of how many times a method increases the number of models available in the market. Only models with an accuracy higher than $1 - e$ is considered usable, in which e is the market Acceptable Error predefined by market managers. To measure the equivalent expansion rate of our framework, we randomly generate 100 different combinations of 5 classes from 5 different original models, where each class is drawn from a different model. These original models are the same as those used in T_3. We plot the relation between the equivalent expansion rate and the market Acceptable Error when $k = 5, 10, 20$ in Figure 6. Our results demonstrate that our framework can effectively expand the market. When we set the market acceptable error to 0.2, the market is amplified to about 200,000 times its original size. Similarly, when we set the market acceptable error to 0.3, the market is expanded to about 600,000 times its original size, representing a significant improvement.

Difference between Fragments. Next, we investigate the differences between each pair of fragments from the same original model in our method. Figure 5 shows the Intersection over Union (IoU) of each layer's selected units set U_i on different fragments in task T_3. We calculate the average IoU among all layers for each pair of fragments in the target of the task. Here, $c_1, c_2, \cdots, c_5$ represent classes in the original model. Our results show that fragments of different classes are significantly different in terms of the selection of units. Compared with selecting units by rate, selecting units by threshold has higher IoU. This is because it prefers to maintain more parameters in the front layers and drop more parameters in the last layers. This observation aligns with the intuition that there are more common sub-abilities in shallow layers of networks. More details are shown in Appendix E.

Running time of Slicing-and-Packing. We show the running time of our method on T_1 using VGG16 and ResNet34 as original models. Figure 7 displays the time spent per class as the number of models required by users increases. As the slope of lines in figure is small, our method only needs little time to pack required fragments, which is insignificant compared with the slicing part. Besides, since the time needed to pack fragments is little, our method has plug-and-play property and user can add or remove fragments at any time.

Conclusion and Future Work

In this paper, we developed a novel framework called Slice-and-Pack. This framework involves slicing existing models into fragments and packing required fragments with necessary abilities, allowing users to purchase models at a lower cost. Additionally, it provides a commercially viable solution for markets and enables them to offer a wider variety of models, demonstrating the capabilities of a larger market. Our experiments have shown that our framework can obtain highly accurate packed models with minimal time spent, particularly on packing, which enables plug-and-play functionality. In our future work, we plan to delve into a more flexible approach to slicing models and leveraging shared capabilities or classes across original models. Moreover, we intend to explore a wider range of network architectures in order to augment our research even further.

Acknowledgments

This research was supported by NSFC (62076121, 61921006) and Major Program (JD) of Hubei Province (2023BAA024). The authors would like to thank Prof. Peng Zhao for his helpful feedback on drafts of the paper.

References

Ben-David, S.; Blitzer, J.; Crammer, K.; and Pereira, F. 2006. Analysis of representations for domain adaptation. *Conference on Neural Information Processing Systems*.

Bengio, Y.; Lecun, Y.; and Hinton, G. 2021. Deep Learning for AI. *Communications of the ACM*.

Ding, Y.-X.; and Zhou, Z.-H. 2020. Boosting-based reliable model reuse. In *Asian Conference on Machine Learning*.

Fernando, B.; Habrard, A.; Sebban, M.; and Tuytelaars, T. 2013. Unsupervised visual domain adaptation using subspace alignment. In *International Conference on Computer Vision*.

Hameed, M. G. A.; Tahaei, M. S.; Mosleh, A.; and Nia, V. P. 2022. Convolutional neural network compression through generalized Kronecker product decomposition. In *AAAI Conference on Artificial Intelligence*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hovy, E.; Gerber, L.; Hermjakob, U.; Lin, C.-Y.; and Ravichandran, D. 2001. Toward Semantics-Based Answer Pinpointing. In *International Conference Human Language Technologies*.

Huang, J.; Gretton, A.; Borgwardt, K.; Schölkopf, B.; and Smola, A. 2006. Correcting sample selection bias by unlabeled data. In *Conference on Neural Information Processing Systems*.

Kim, Y. 2014. Convolutional Neural Networks for Sentence Classification. *arXiv:1408.5882*.

Kim, Y.-D.; Park, E.; Yoo, S.; Choi, T.; Yang, L.; and Shin, D. 2015. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*.

Krizhevsky, A.; Hinton, G.; et al. 2009. Learning multiple layers of features from tiny images. *Toronto, ON, Canada*.

Kuzborskij, I.; and Orabona, F. 2013. Stability and hypothesis transfer learning. In *International Conference on Machine Learning*.

Kuzborskij, I.; and Orabona, F. 2017. Fast rates by transferring from auxiliary hypotheses. *Machine Learning*.

Li, T.; Li, J.; Liu, Z.; and Zhang, C. 2020a. Few sample knowledge distillation for efficient network compression. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Li, Y.; Gu, S.; Mayer, C.; Gool, L. V.; and Timofte, R. 2020b. Group sparsity: The hinge between filter pruning and decomposition for network compression. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Liang, P.; Bommasani, R.; Lee, T.; Tsipras, D.; Soylu, D.; Yasunaga, M.; Zhang, Y.; Narayanan, D.; Wu, Y.; Kumar, A.; et al. 2022. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*.

Luo, R.; Sun, L.; Xia, Y.; Qin, T.; Zhang, S.; Poon, H.; and Liu, T.-Y. 2022. BioGPT: generative pre-trained transformer for biomedical text generation and mining. *Briefings in Bioinformatics*.

Nguyen, D.; Nguyen, K.; Ho, N.; Phung, D.; and Bui, H. 2021. Model Fusion of Heterogeneous Neural Networks via Cross-Layer Alignment. *arXiv preprint arXiv:2110.15538*.

OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774.

Pan, S. J.; Tsang, I. W.; Kwok, J. T.; and Yang, Q. 2010. Domain adaptation via transfer component analysis. *IEEE Transactions on Neural Networks and Learning Systems*.

Pan, S. J.; and Yang, Q. 2010. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*.

Pathak, M.; Rane, S.; and Raj, B. 2010. Multiparty differential privacy via aggregation of locally trained classifiers. *Conference on Neural Information Processing Systems*.

Radford, A.; Kim, J. W.; Hallacy, C.; Ramesh, A.; Goh, G.; Agarwal, S.; Sastry, G.; Askell, A.; Mishkin, P.; Clark, J.; et al. 2021. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*.

Shen, C.; Wang, X.; Yin, Y.; Song, J.; Luo, S.; and Song, M. 2021. Progressive network grafting for few-shot knowledge distillation. In *AAAI Conference on Artificial Intelligence*.

Shu, Y.; Kou, Z.; Cao, Z.; Wang, J.; and Long, M. 2021. Zoo-tuning: Adaptive transfer from a zoo of models. In *ICML*.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature*.

Simonyan, K.; and Zisserman, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Socher, R.; Perelygin, A.; Wu, J.; Chuang, J.; Manning, C. D.; Ng, A. Y.; and Potts, C. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Conference on Empirical Methods in Natural Language Processing*.

Sun, H.; Xie, Z.; Li, X.-Y.; and Li, M. 2023. Cooperative and Adversarial Learning: Co-Enhancing Discriminability and Transferability in Domain Adaptation. In *AAAI Conference on Artificial Intelligence*.

Tan, P.; Tan, Z.-H.; Jiang, Y.; and Zhou, Z.-H. 2022. Towards enabling learnware to handle heterogeneous feature spaces. *Machine Learning*.

Wu, X.-Z.; Liu, S.; and Zhou, Z.-H. 2019. Heterogeneous model reuse via optimizing multiparty multiclass margin. In *International Conference on Machine Learning*.

Wu, X.-Z.; Xu, W.; Liu, S.; and Zhou, Z.-H. 2021. Model reuse with reduced kernel mean embedding specification. *IEEE Transactions on Knowledge and Data Engineering*.

Yang, X.; Ye, J.; and Wang, X. 2022. Factorizing knowledge in neural networks. In *European Conference on Computer Vision*.

Yang, X.; Zhou, D.; Liu, S.; Ye, J.; and Wang, X. 2022. Deep model reassembly. In *Conference on Neural Information Processing Systems*.

Yang, Y.; Shen, H. T.; Ma, Z.; Huang, Z.; and Zhou, X. 2011. $\ell_{2,1}$-norm regularized discriminative feature selection for unsupervised learning. In *International Joint Conference on Artificial Intelligence*.

Zhang, F.; Zhu, X.; and Ye, M. 2019. Fast human pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Zhang, H.; Chen, D.; and Wang, C. 2022. Confidence-aware multi-teacher knowledge distillation. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*.

Zhang, X.; Zou, J.; He, K.; and Sun, J. 2015. Accelerating very deep convolutional networks for classification and detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Zhao, P.; Cai, L.-W.; and Zhou, Z.-H. 2020. Handling concept drift via model reuse. *Machine Learning*.

Zhou, Z. 2016. Learnware: on the future of machine learning. *Frontiers of Computer Science*.

Zhou, Z.; and Tan, Z. 2022. Learnware: Small Models Do Big. *CoRR*.