Algorithm 1 Module-mixing Model.

```
Data: Training data $D_t = \{\mathcal{X}_t, \mathcal{Y}_t\}$; Selected source models $\{\boldsymbol{f}_{s_n}\}_{n \in S_m}$;
Result: White-box: Module-mixing model $\boldsymbol{f}$ constructed with (4) and (3); Black-box: Module-mixing model $\boldsymbol{f}$ constructed
        with only (4)
for $epoch \leftarrow 1$ to $C$ do
    for each minibatch $\boldsymbol{x}$ do
        for $n \leftarrow 1$ to $S_m$ do
            Calculate distance matrices $\boldsymbol{A}(\boldsymbol{f}_{s_n}; \boldsymbol{x})$ and $\boldsymbol{B}(\boldsymbol{f}_t; \boldsymbol{x})$ in equation (5)
            Calculate $\mathcal{L}_{DC}(\boldsymbol{f}_{s_n}(\boldsymbol{x}), \boldsymbol{f}_t(\boldsymbol{x}))$ as in (5)
        end
        Calculate $\mathcal{L}_{total}$ as in (6)
        Update $\boldsymbol{\theta}_{new}, \boldsymbol{c}_t, \boldsymbol{\lambda}_t$ using $\boldsymbol{\theta}^*_{new}, \boldsymbol{c}^*_t, \boldsymbol{\lambda}^*_t = \arg\min \mathcal{L}_{total}(\boldsymbol{h}_t; \mathcal{X}_t, \mathcal{Y}_t)$
    end
end
```

A. EFFICIENT FEATURE TRANSFORMATION

[4] proposed efficient feature transformation (EFT) for both 2D convolutional layers and fully connected layers. In the experiments, we apply EFT to convolutional layers in a ResNet18 model. As mentioned in the Background section, given the groupwise convolutional kernels $\omega^s \in \mathbb{R}^{M \times Q \times a}$ and the corresponding $\frac{K}{a}$ groups of feature maps F, each group of feature maps $F^s_{ai:(ai+a-1)}$ is convolved with kernels $[\omega^s_{i,1}, \cdots, \omega^s_{i,a}]$ to form a new group of feature maps $H^s_{ai:(ai+a-1)} \in \mathbb{R}^{M \times Q \times a}$. The formulation of which is shown below:

$$\begin{aligned} H^s_{ai:(ai+a-1)} = [\omega^s_{i,1} * F_{ai:(ai+a-1)} | \cdots | \\ \omega^s_{i,a} * F_{ai:(ai+a-1)}], i \in \{0, \cdots, \frac{K}{a} - 1\}, \end{aligned} \tag{7}$$

where $|$ is the concatenation operation and i indicates the group of feature maps. Then, all the $H^s_{ai:(ai+a-1)}$ are concatenated to form H^s. Same procedures can be applied with the pointwise convolutional kernels $\omega^d \in \mathbb{R}^{M \times Q \times b}$ and the corresponding $\frac{K}{b}$ groups of feature maps F:

$$\begin{aligned} H^s_{bi:(bi+b-1)} = [\omega^s_{i,1} * F_{bi:(bi+b-1)} | \cdots | \\ \omega^s_{i,b} ss * F_{bi:(bi+b-1)}], i \in \{0, \cdots, \frac{K}{b} - 1\}. \end{aligned} \tag{8}$$

Then concatenate $H^d_{ai:(ai+a-1)}$ to form H^d. Combining H^d and H^s produces the final feature maps $H = H^s + \gamma H^d$, where $\gamma \in \{0, 1\}$, and in this paper we set $\gamma = 1$.

By setting $a \ll K$ and $b \ll K$, the amount of trainable parameters per task is substantially reduced. For instance, using a ResNet18 backbone, $a = 8$, and $b = 16$ results in 449k parameters per new task, which is 3.9% the size of the backbone. As empirically demonstrated in [4], EFT makes model growth efficient in continual learning settings while preserving the remarkable representation power of ResNet.

B. ALGORITHM

The pseudo code for training our module-mixing model is shown in Algorithm 1.

C. DATASET AND IMPLEMENTATION DETAILS

Hyperparamter settings For experiments in the *white-box* setting, *i)* with `Office-31`, all the models are trained for 100 epochs with batch size 128; the training starts with a learning rate of 0.001 and weight decay $1e^{-5}$, and the learning rate decays by 0.1 after the first 50 epochs. *ii)* For `CIFAR100`, all the models are trained for 150 epochs with batch size 128, weight decay of $1e^{-5}$, and a learning rate starts at 0.01 and decays by 0.1 after 100 epochs, except for the tasks with dataset size 40, which is trained with weight decay $1e^{-4}$. For tasks with dataset size 40, we also add random horizontal flip data augmentation to all the experiments to further reduce overfitting. *iii)* For `CtrL`, all the experiments are trained for 100 epochs with batch size 16. We perform a grid search on learning rate in $\{0, 0.01, 0.003\}$ and weight decay in $\{0, 1e^{-4}, 1e^{-5}\}$, and compare the results to having the same learning rate of 0.01 and reduced by 0.1 after 50 epochs, with weight decay $1e^{-4}$. We also add random horizontal flip augmentation, as in the original paper [11], to alleviate overfitting.

For experiments in the *black-box* setting, the learning rate is set to 0.001 and reduced by 0.1 after 50 epochs, batch size is 128, and weight decay is $1e^{-5}$ for all datasets. We adopt the scikit-learn package implementation of FastICA to reduce the dimensionality of the feature outputs obtained from the source models.

Mixing weights settings In Tables 1, 6, 2, and 9, uniform initialization is set for the mixing weights in each layer. In the Additional Experimental Results section, results with different mixing weight initialization on `Office-31` data can be found.

C.1. Details on CIFAR100

`CIFAR100` has 100 classes containing 600 images per class, which is further split into 500 training images and 100 test images for each class. We construct a list of 40 tasks $[1, 2, \cdots, 40]$ as in Table 11 by evenly splitting each class in `CIFAR100` into 300 images, which means each class in the 40 tasks has 250 training images and 50 test images. For each task, we randomly select a subset of size $e = \{40, 80, 160, 320\}$ out of the training images to train the module-mixing model, with 10% of the selected training images observed as a validation set. However, for training with different subsets, the test set remains the same, which contains 250 images.

C.2. Details on CTrL

Each task in the S_{long} dataset consists of training, validation, and test datasets. All images have been rescaled to 32x32 pixels in RGB color format, and normalized per-channel using statistics computed on the training set of each task. Data augmentation is performed by using random crops (4 pixels padding and 32x32 crops) and random horizontal flips.

The details of the tasks used in this paper can be found in Tables 12 and 13. Note that though we used the same random seed (343008097) to generate the dataset as in the original paper, the generated tasks with our hardware have different classes and dataset sizes from the original paper [11].

C.3. Details on Office-31

`Office-31` consists of 31 categories of images originating from 3 domains: Amazon, DSLR and Webcam, including common objects in everyday office settings. Webcam consists of 795 low-resolution images with noise; the Amazon domain has 2817 images from online merchants; DSLR images are captured with a digital camera, and all 423 images are of high resolution and low noise. For each domain in Office31, we randomly select 80% of its data as training set, 10% as validation set, and 10% as test set. Data augmentation is performed by using random crops and random horizontal flips. All images have been rescaled to 256×256 and then cropped to 224×224.

Hyperparameter grid search In the experiment shown in Table 6, learning rates $\{10^{-2}, 10^{-3}\}$, weight decay strengths $\{0, 10^{-5}, 10^{-4}\}$ and weight initializations $\lambda_{new} \in \{0.001, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.997\}$ are considered (0.001 and 0.997 are used to avoid setting weights to zero). For each λ_{new}, other mixing weights in the same layer are set to a same value $(1 - \lambda_{new})/m$, with m being the number of selected source models. We select the hyperparameter combination that produces the best validation accuracy.

D. ADDITIONAL EXPERIMENTAL RESULTS

D.1. White-box Scenario

Office-31 We show the results with different mixing weight initialization in Table 3. λ_{new} is initialized with $[0.01, 0.4, 0.5, 0.6, 0.99]$ for both $m = 1$ and $m = 2$.

Table 3: `Office-31` in White-box scenario with different mixing weight initialization.

Method	$A, D \rightarrow W$	$A, W \rightarrow D$	$W, D \rightarrow A$	AVG
Independent model	69.61	75.62	71.73	72.34
Finetune Source	91.25	88.24	82.69	87.39
m=1 $\lambda_{new} = 0.01$	78.25	76.47	94.35	83.02
m=1 $\lambda_{new} = 0.4$	96.25	90.20	92.58	**93.01**
m=1 $\lambda_{new} = 0.5$	**97.50**	88.24	91.52	92.42
m=1 $\lambda_{new} = 0.6$	95.00	88.24	85.87	89.70
m=1 $\lambda_{new} = 0.99$	92.50	94.12	68.20	84.94
m=2 $\lambda_{new} = 0.01$	81.25	64.71	**96.47**	80.81
m=2 $\lambda_{new} = 0.4$	93.75	92.16	90.81	92.24
m=2 $\lambda_{new} = 0.5$	92.50	**94.12**	85.16	90.59
m=2 $\lambda_{new} = 0.6$	96.26	92.16	83.75	90.72
m=2 $\lambda_{new} = 0.99$	93.75	90.20	74.56	86.17

CIFAR100 For a closer look at the results shown in Figures 3, we also present the results in Tables 4 and 5, respectively.

Table 4: Influence of the number of source models on performance for `CIFAR100`.

	320	160	80	40
Independent model	63.44 ± 0.54	52.29 ± 0.61	50.79 ± 0.79	44.69 ± 0.93
Finetune Source	62.99 ± 0.60	55.81 ± 0.54	49.47 ± 0.80	41.68 ± 0.99
$m = 1$	63.47 ± 0.55	**56.20** ± 0.78	51.56 ± 0.72	**47.01** ± 0.99
$m = 3$	63.44 ± 0.50	55.85 ± 0.63	52.05 ± 0.65	46.19 ± 0.94
$m = 5$	**63.83** ± 0.43	55.27 ± 0.55	**52.67** ± 0.76	46.49 ± 1.43
AVG	63.58	55.77	52.09	46.56

CTrL With this challenging dataset, we have a large pool of models from which to start, and source and target tasks come from very distinct datasets and have diverse sample sizes. The S^{long} stream from `CTrL` is a collection of 100 tasks created from five well-known computer vision datasets: `CIFAR10`, `CIFAR100`, `SVHN` , `MNIST`, and `Fashion-MNIST` (short as `FMNIST`). Each task in S^{long} is constructed by first randomly selecting a dataset, then five classes of the chosen dataset, and finally, a large task (containing 5000 training samples) or a small task (containing 25 training samples). During tasks $1 - 33$, the fraction of small tasks is 50%; this increases to 75% for tasks $34 - 66$, and to 100% for tasks $67 - 100$. More details of the dataset can be found in the Appendix. In our experiments, we use the first 60 tasks as our starting pool of source models and gradually add newly trained models into the collection. We report the average test accuracy on the last 40 tasks. This is a challenging problem not only because it simulates a realistic setting that starts with a large collection of pre-trained tasks for repurposing, but also because the last 40 tasks have only 25 training samples each, which could cause serious overfitting problems. We perform a simple grid search on hyperparameters (learning rate and weight decay) and compare them to having the same settings for all tasks. Grid search and other experimental details are in the Appendix. Results show that grid search is necessary when tasks come from very different datasets. In Table 6, we also show results of an extension of MCW and multi-source SVM with our k-NN source model sampler. The results show the effectiveness of our k-NN source model sampler, helping to achieve a 23.52% increase for MCW compared to learning from randomly sampled source models. However, we

Table 5: Influence of different mixing weight initialization on `CIFAR100` when $m = 5$.

	320	160	80	40
Independent model	63.44 ± 0.54	52.29 ± 0.61	50.79 ± 0.79	44.69 ± 0.93
$\lambda_{new} = 0.16$	63.83 ± 0.43	55.27 ± 0.55	**52.67** ± 0.76	**46.49** ± 1.43
$\lambda_{new} = 0.3$	63.84 ± 0.56	**56.43** ± 0.72	51.55 ± 0.67	45.52 ± 1.22
$\lambda_{new} = 0.4$	63.91 ± 0.62	53.28 ± 0.66	51.15 ± 0.77	45.35 ± 0.99
$\lambda_{new} = 0.6$	**64.20** ± 0.58	52.68 ± 0.51	50.98 ± 0.81	44.87 ± 1.15
AVG	63.95	54.42	51.59	45.58

Table 6: `CTrL` in the white-box scenario.

Method	$m = 1$	$m = 3$	$m = 5$	AVG
Cross-dataset soup	47.73	43.83	39.43	43.66
MCW	42.73	45.02	54.09	47.28
Independent	-	-	-	45.77
Finetune Source	-	-	-	54.26
MCW (knn)	66.59	**72.89**	**72.91**	**70.80**
Ours (w/o $\mathcal{L}_{DC}$)	56.40	55.13	52.09	54.54
Ours (w/$\mathcal{L}_{DC}$)	56.48	55.35	52.79	54.87
Ours (w/$\mathcal{L}_{DC}$ grid search)	67.26	70.24	68.71	68.73

still suffer from the overfitting problem mainly on tasks created with `SVHN`, which explains the gap between our method and MCW with k-NN sampler.

TinyImageNet To show that our method also applies to more complex tasks, we provide an experiment on `Tiny-ImageNet`. We use a randomly initialized ResNet-18 as the backbone. The purpose of this setting is to show that the improvement in performance of our model does not rely on a powerful and closely related backbone model since `Tiny-ImageNet` is a subset of `ImageNet`. We create 6 tasks with 200 classes in the dataset, with 50 classes for each task. Task 1 through 5 have overlapping classes, while Task 6 has no overlapping with other tasks. Task details are presented in Table 14 and 15. Since the tasks in this setting all have sufficient training data, the newly initialized task-specific modules for the target task are given higher initial module-mixing weights (0.9) to encourage the learning of target task-specific knowledge. For each task, we search for top-m most similar tasks in the other 5 tasks. In Table 8, we also provide the task selection results. We observe that tasks with overlapping classes are always picked for each task.

D.2. Black-box Scenario

In Table 7, we also provide results on `CTrL` under black-box setting for source models. A LeNet-5 model is used as target model. Random horizontal flip and vertical flip data augmentation is used to increase the size of training set to 150 and validation set to 90, a batch size of 128 is used during training in this setting.

Table 7: `CTrL` in black-box scenario.

Method	$m = 1$	$m = 3$	$m = 5$	API	AVG
Independent	-	-	-	-	45.14
Finetune Source	-	-	-	-	46.31
Ours (w/o $\mathcal{L}_{DC}$)	47.12	47.23	**47.42**	46.40	**47.04**
Ours (w/ $\mathcal{L}_{DC}$)	**47.33**	**47.26**	46.83	**46.41**	46.96

Table 8: `TinyImageNet` in the white-box scenario.

	Task1	Task2	Task3	Task4	Task5	Task6
Model stacking	31.03	31.86	35.33	31.43	30.64	28.91
Cross-dataset soup	28.65	32.20	**37.24**	**34.20**	33.51	32.73
Multi-source SVM	28.64	25.44	25.76	24.72	27.04	23.76
MCW	21.76	22.32	22.40	21.68	22.16	21.68
Independent model	34.55	32.64	35.07	32.47	33.16	33.25
Finetune source	34.38	31.77	33.51	32.73	33.59	32.90
Ours $m = 1$	35.16	33.59	36.26	32.90	**35.07**	**35.42**
Selected task	Task2	Task3	Task2	Task5	Task4	Task4
Ours $m = 2$	**36.40**	**34.08**	33.68	33.52	34.96	33.84
Selected tasks	Task2	Task1	Task2	Task3	Task4	Task1
	Task3	Task3	Task4	Task5	Task6	Task4

D.3. UMAP Plots of Office-31 Domains

Here we provide additional UMAP plots on DSLR and Amazon domains. For the DSLR domain, the same conclusions can be drawn as for the Webcam domain, *i.e.*, the learned features with our model are relatively more clustered and separated compared to that for the other two methods as shown in Figure 6. As for the Amazon domain, in Figure 7, we create scatter plots of the learned UMAP embeddings of the training data, and color samples (points) according to their ground truth labels. All three plots in Figure 7 have good clustering patterns. However, if we color-code the training data points by their respective predicted probabilities of outcomes as in Figure 8, we notice that models Independent and Finetune Source are overconfident about their predictions, while our method also has the effect of preventing the model from predicting the labels too confidently during training and thus improving generalization. In Figure 9, we visualize the extracted features of the test data after UMAP embedding. We see that the Independent model is less confident about the predictions since the model does not generalize well. While Finetune Source has higher predicted probabilities on the test set, our model has relatively better clustering and is more certain about the predictions compared to Finetune Source.

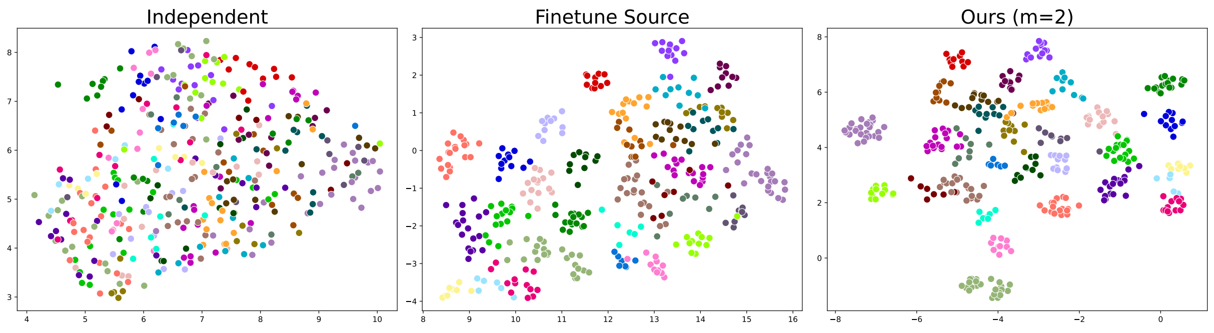

Fig. 6: Umap plots on the DSLR domain with training data. Each unique color represents the same label across all three plots.

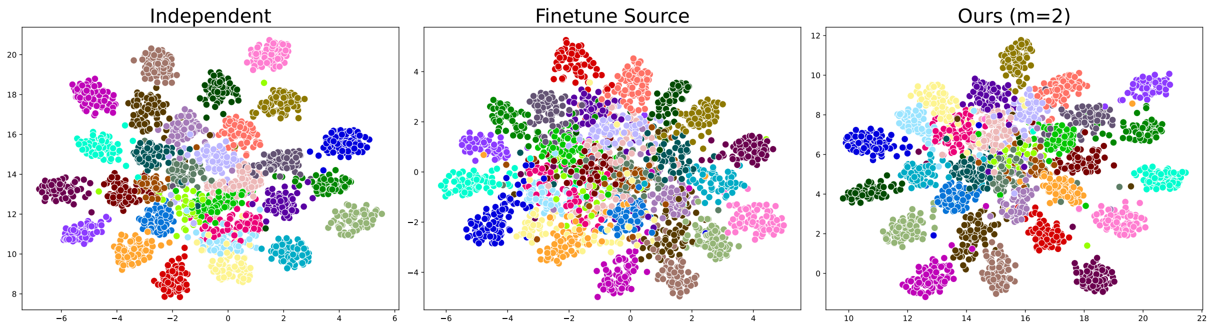

Fig. 7: Umap plots on the Amazon Domain with training data. Each unique color represents the same label across all three plots.

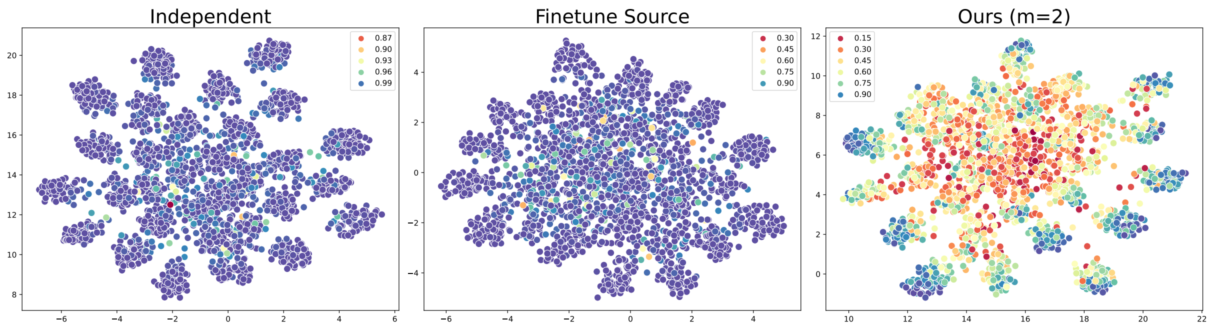

Fig. 8: Umap plots on the Amazon domain with training data. The colors represent the predicted probability.

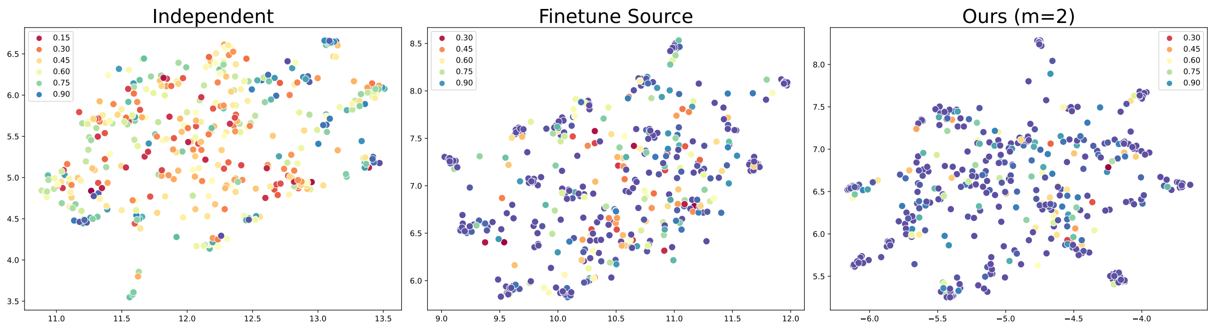

Fig. 9: Umap plots on the Amazon domain with testing data. The colors represent the predicted probability.

D.4. Ablation Study

We construct a module-mixing model with the top-1 source model selected via k-NN and a target model using only (4). In this setting, the mixing weight λ_1 for the source model features is set as a value in $\{0, 0.1, \ldots, 0.9, 1\}$, and only the target model's modules and the classification head are trained. Thus, an independent model is trained when $\lambda_1 = 0$, whereas only the classification head of a pre-trained source model is trained when $\lambda_1 = 1$. We test on our 30 CIFAR100 tasks. For each task, one setting for λ_1 will achieve the highest test accuracy. Each bar in Figure 10 (Right) shows the percentage of tasks that reached peak accuracy at the given task weight setting. We see that the optimum results are mostly obtained (93%) when the mixing weight is non-zero for either source and target models, thus justifying our model design. In Table 9, we examine the advantages of our source model selection and layer-wise module-mixing design on CTrL (with $m = 1$ for adaptation). Alternatively, we consider selecting source models at random (instead of k-NN) and only using the convex combination of features in (4) (instead of both features and module parameters). We see that with random selection and only feature aggregation, we get an

Table 9: Ablation Study on `CTrL` dataset when $m = 1$.

Method	Accuracy
Independent model	45.77
Random (only (4))	45.88
k-nearest neighbors (only 4)	55.68
Random (both (4) and (3))	46.61
k-nearest neighbors (both (4) and (3))	**56.40**

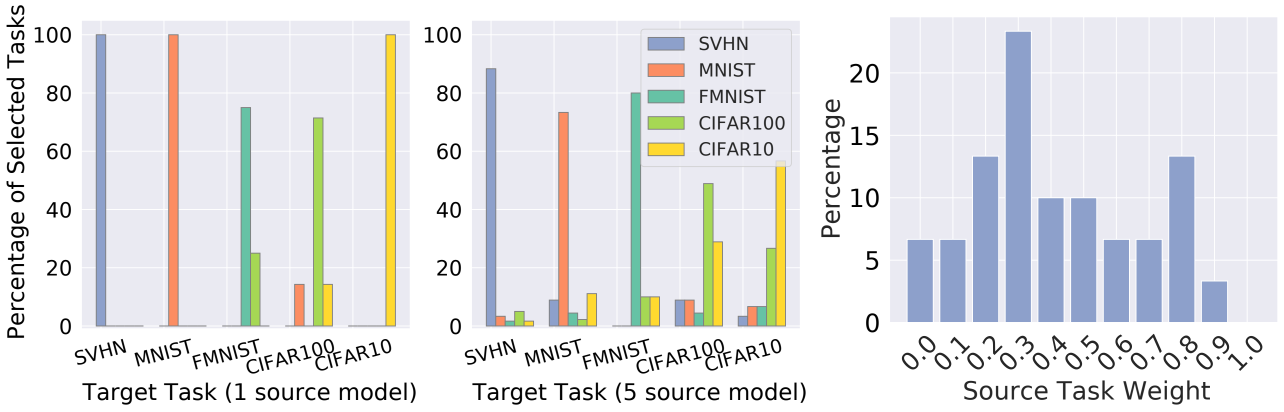

Fig. 10: Selection of $m = 1$ (Left) and $m = 5$ (Middle) related tasks via k-NN. The x-axis shows the membership of the target tasks, while the y-axis shows the proportion of selected source tasks for each membership. Right: Module-mixing results for $m = 1$. The x-axis is the mixing weight value set for the source model and the y-axis is the percentage of tasks in `CIFAR100` for which the highest accuracy is obtained at a given source weight setting.

insignificant gain of 0.11% relative to the independent model, while with k-NN selection and only feature aggregation, the gain is higher at 1%. Importantly, the complete approach using (4) and (3) yields a substantial 9.79% gain.

Moreover, we show how well the k-NN algorithm selects source models with `CTrL`. In Figure 10, we show selection results with $m = 1$ (Left) and $m = 5$ (Middle). Each bar represents the proportion of selected tasks from each dataset for the 40 tasks. When $m = 1$, target tasks created with `SVHN`, `MNIST`, and `CIFAR10` always pick source models trained from the same datasets. Also, we only have 4 `FMNIST` tasks in the experiments, which explains why the selector picks a high 25% of `CIFAR10` source models. As m increases to 5, tasks with classes from `SVHN`, `MNIST`, `FMNIST` still almost always pick models created from the same dataset. For `CIFAR10` and `CIFAR100`, since they both contain images of natural objects, they are more similar.

D.5. Ablation Study on the a/b Parameter Settings for EFT

a=8 and b=16 are the optimal parameter choices according to the original EFT paper [4] for `CIFAR100`. We adopted them for our experiments on `CIFAR100` and `Office-31`, where the number of samples for each task is relatively sufficient.

We set $a = 2$ and $b = 1$ for `CtrL` to reduce the overfitting problem on training source models with limited data since most of the source tasks have very limited data, we reduce the number of training parameters accordingly.

We also tried different a/b settings for `CtrL`, and $a = 2/b = 1$ gives the best performance for independent models. Below are some examples for training independent models with different a/b settings on tasks 60 to 100, which all have a limited data size. Weight decay is set to $1e-4$ and learning rate 0.01 for each setting, and all the experiments are run with the same three random seeds.

D.6. Study on the Setting of k for k-NN

We study how different choices of k for k-NN algorithm influence the selection results for source models using the `CTrL` dataset. We plot the membership selection results in Figure 11. We can see that the selected source models are more related to

Table 10: Influence of the number of source models on performance for `CIFAR100`.

	a=1/b=1	a=2/b=1	a=1/b=2	a=2/b=2
Average test accuracy with three random seeds	43.2/44.8/43.9	47.2/46.1/45.9	44.4/45.3/44.9	46.1/45.5/47.2
Avg of three runs	44.0	**46.4**	44.9	46.3

the target tasks with a larger value for k.

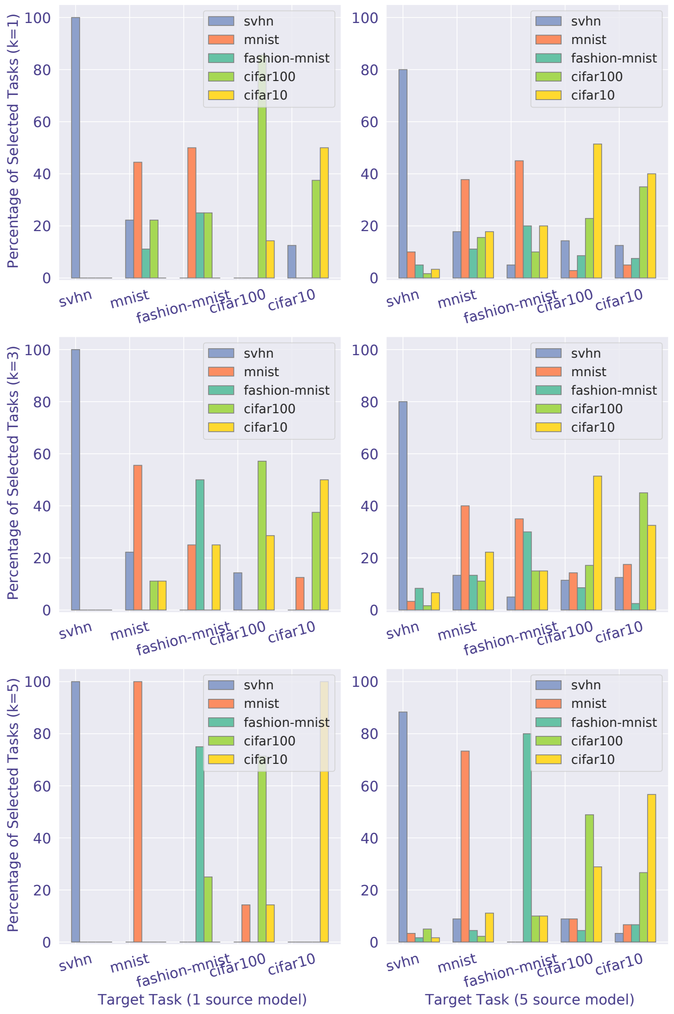

Fig. 11: Study on the different setting of k for k-NN source model selection. From top to bottom, the results are for when $k = 1$, $k = 3$ and $k = 5$.

Table 11: The 40 Tasks created with `CIFAR100`.

Task ID	Classes
1	[aquarium_fish, flatfish, ray, shark, trout]
2	[orchid, poppy, rose, sunflower, tulip]
3	[beaver, dolphin, otter, seal, whale]
4	[bottle, bowl, can, cup, plate]
5	[bear, leopard, lion, tiger, wolf]
6	[apple, mushroom, orange, pear, sweet_pepper]
7	[hamster, mouse, rabbit, shrew, squirrel]
8	[baby, boy, girl, man, woman]
9	[maple_tree, oak_tree, palm_tree, pine_tree, willow_tree]
10	[bicycle, bus, motorcycle, pickup_truck, train]
11	[clock, keyboard, lamp, telephone, television]
12	[bed, chair, couch, table, wardrobe]
13	[bee, beetle, butterfly, caterpillar, cockroach]
14	[bridge, castle, house, road, skyscraper]
15	[cloud, forest, mountain, plain, sea]
16	[camel, cattle, chimpanzee, elephant, kangaroo]
17	[fox, porcupine, possum, raccoon, skunk]
18	[crab, lobster, snail, spider, worm]
19	[crocodile, dinosaur, lizard, snake, turtle]
20	[lawn_mower, rocket, streetcar, tank, tractor]
21	[hamster, cattle, maple_tree, squirrel, chimpanzee]
22	[house, bridge, bicycle, baby, lamp]
23	[shark, oak_tree, shrew, beaver, plate]
24	[willow_tree, crocodile, tiger, otter, telephone]
25	[bus, aquarium_fish, camel, skunk, apple]
26	[bee, forest, tank, sweet_pepper, fox]
27	[snail, whale, clock, lion, cockroach]
28	[rabbit, castle, pine_tree, cloud, boy]
29	[butterfly, road, rocket, skyscraper, wardrobe]
30	[spider, crab, tractor, mouse, seal]
31	[bed, elephant, beetle, keyboard, train]
32	[plain, table, ray, worm, sea]
33	[trout, bear, kangaroo, caterpillar, turtle]
34	[motorcycle, bottle, orchid, chair, leopard]
35	[dolphin, can, porcupine, cup, pickup_truck]
36	[poppy, wolf, pear, bowl, man]
37	[snake, tulip, streetwar, palm_tree, girl]
38	[lawn_mower, television, mushroom, lizard, raccoon]
39	[orange, dinosaur, possum, lobster, flatfish]
40	[mountain, couch, rose, woman, sunflower]

Table 12: Details of CTrL S_{long} (tasks 1-50).

Task ID	Dataset	Classes	#Train	# Val	# Test
1	mnsit	[3, 0, 5, 6, 8]	5000	2500	4804
2	svhn	[9, 1, 3, 7, 0]	25	15	5000
3	svhn	[2, 9, 7, 1, 3]	25	15	5000
4	svhn	[5, 3, 0, 2, 9]	5000	2500	5000
5	fashion-mnist	[Sandal, Dress, T-shirt/top, Ankle boot, Pullover]	25	15	5000
6	fashion-mnist	[Ankle boot, Sneaker, Dress, Shirt, Trouser]	25	15	5000
7	svhn	[5, 4, 6, 3, 7]	5000	2500	5000
8	cifar100	[man, squirrel, mouse, keyboard, maple_tree]	2250	250	500
9	cifar10	[horse, cat, bird, frog, dog]	5000	2500	5000
10	fashion-mnist	[Coat, Ankle boot, Shirt, Pullover, Sneaker]	5000	2500	5000
11	mnist	[0, 7, 1, 9, 6]	25	15	4938
12	cifar10	[bird, ship, airplane, dog, frog]	5000	2500	5000
13	cifar100	[hamster, bear, dolphin, bicycle, road]	25	15	500
14	mnist	[9, 5, 0, 8, 3]	5000	2500	4846
15	fashion-mnist	[Sandal, Trouser, Shirt, Bag, Coat]	5000	2500	5000
16	cifar10	[frog, bird, deer, automobile, horse]	5000	2500	5000
17	cifar10	[deer, ship, truck, airplane, frog]	5000	2500	5000
18	svhn	[7, 0, 6, 5, 3]	5000	2500	5000
19	mnist	[8, 5, 7, 6, 4]	25	15	4806
20	mnist	[1, 4, 2, 0, 9]	25	15	4962
21	cifar100	[whale, train, boy, crab, caterpillar]	2250	250	500
22	cifar100	[rabbit, cattle, camel, cockroach, caterpillar]	2250	250	500
23	cifar100	[orchid, road, spider, snake, caterpillar]	25	15	500
24	svhn	[3, 7, 8, 4, 5]	25	15	5000
25	cifar100	[cloud, raccoon, baby, lamp, orange]	2250	250	500
26	cifar100	[dolphin, castle, flatfish, house, whale]	2250	250	500
27	cifar100	[skunk, chimpanzee, tractor, raccoon, lion]	25	15	500
28	svhn	[8, 6, 9, 2, 5]	5000	2500	5000
29	fashion-mnist	[Coat, Sandal, Bag, Ankle boot, Trouser]	5000	2500	5000
30	mnist	[9, 8, 2, 6, 7]	5000	2500	4932
31	cifar10	[truck, ship, deer, bird, automobile]	25	15	5000
32	cifar10	[airplane, horse, dog, bird, cat]	25	15	5000
33	svhn	[7, 8, 4, 5, 2]	25	15	5000
34	cifar100	[castle, fox, shark, skunk, crocodile]	2250	250	500
35	cifar100	[wardrobe, bridge, otter, lawn_mower, telephone]	25	15	500
36	fashion-mnist	[Bag, Coat, T-shirt/top, Dress, Sandal]	25	15	5000
37	cifar10	[deer, airplane, automobile, dog, cat]	25	15	5000
38	cifar10	[cat, deer, frog, horse, truck]	5000	2500	5000
39	svhn	[0, 7, 8, 4, 3]	25	15	5000
40	mnist	[5, 7, 9, 2, 4]	5000	2500	4874
41	cifar100	[wolf, dinosaur, ray, girl, crab]	2250	250	500
42	fashion-mnist	[Sandal, Coat, Sneaker, Trouser, Dress]	25	15	5000
43	cifar100	[snail, rose, sweet_pepper, worm, mushroom]	25	15	500
44	fashion-mnist	[Sneaker, Trouser, Tshirt/top, Dress, Pullover]	25	15	5000
45	fashion-mnist	[Bag, Sandal, Shirt, Sneaker, Dress]	25	15	5000
46	mnist	[1, 4, 9, 6, 8]	25	15	4914
47	cifar10	[frog, dog, truck, horse, bird]	25	15	5000
48	svhn	[0, 2, 4, 1, 7]	5000	2500	5000
49	cifar100	[bed, tiger, lamp, road, caterpillar]	2250	250	500
50	cifar100	[rocket, mouse, butterfly, road, cattle]	2250	250	500

Table 13: Details of `CTrL` S_{long} (tasks 51-100).

Task ID	Dataset	Classes	#Train	# Val	# Test
51	cifar100	[trout, road, plain, television, wardrobe]	25	15	500
52	cifar100	[ray, pear, lion, whale, wardrobe]	25	15	500
53	cifar10	[bird, dog, deer, frog, truck]	25	15	5000
54	mnist	[0, 7, 4, 9, 6]	25	15	4920
55	fashion-mnist	[Shirt, Sneaker, Pullover, Trouser, Ankle boot]	5000	2500	5000
56	fashion-mnist	[Bag, Pullover, Sneaker, Tshirt/top, Shirt]	25	15	5000
57	mnist	[7, 9, 3, 8, 0]	25	15	4954
58	cifar10	[ship, airplane, frog, automobile, deer]	25	15	5000
59	mnist	[5, 6, 2, 4, 7]	25	15	4832
60	mnist	[5, 4, 2, 6, 0]	25	15	4812
61	svhn	[0, 2, 7, 3, 6]	25	15	5000
62	mnist	[9, 8, 7, 0, 4]	25	15	4936
63	cifar10	[frog, horse, airplane, ship, automobile]	25	15	5000
64	mnist	[9, 1, 2, 8, 6]	25	15	4932
65	svhn	[7, 2, 5, 9, 1]	25	15	5000
66	cifar100	[streetcar, beaver, sea, dolphin, butterfly]	25	15	500
67	svhn	[6, 3, 9, 7, 2]	25	75	5000
68	mnist	[9, 5, 7, 0, 6]	25	15	4830
69	fashion-mnist	[Tshirt/top, Bag, Dress, Shirt, Sneaker]	25	15	5000
70	svhn	[8, 9, 4, 1, 6]	25	15	5000
71	svhn	[5, 1, 0, 7, 2]	25	15	5000
72	mnist	[5, 1, 4, 2, 7]	25	15	4874
73	cifar10	[frog, cat, horse, truck, deer]	25	15	5000
74	mnist	[3, 9, 4, 2, 8]	25	15	4956
75	cifar10	[ship, truck, deer, dog, bird]	25	15	5000
76	cifar100	[pear, bus, fox, cloud, oak_tree]	25	15	500
77	svhn	[1, 0, 2, 3, 7]	25	15	5000
78	svhn	[2, 8, 5, 4, 3]	25	15	5000
79	cifar100	[rabbit, woman, girl, skyscraper, tuplip]	25	15	500
80	cifar100	[trout, road, bridge, bowl, oak_tree]	25	15	500
81	mnist	[5, 4, 8, 0, 6]	25	15	4786
82	fashion-mnist	[Pullover, Dress, Coat, Bag, Tshirt/top]	25	15	5000
83	cifar10	[bird, cat, deer, dog, automobile]	25	15	5000
84	cifar10	[cat, dog, ship, frog, bird]	25	15	5000
85	cifar100	[chimpanzee, camel, palm_tree, leopard, spider]	25	15	500
86	mnist	[3, 4, 5, 9, 6]	25	15	4832
87	svhn	[8, 9, 4, 7, 2]	25	15	5000
88	cifar10	[horse, cat, dog, airplane, automobile]	25	15	5000
89	mnist	[9, 8, 0, 1, 6]	25	15	4912
90	cifar100	[possum, chair, bowl, mountain, cloud]	25	15	500
91	svhn	[7, 1, 5, 9, 6]	25	15	5000
92	cifar10	[bird, cat, ship, frog, deer]	25	15	5000
93	mnist	[2, 8, 7, 5, 6]	25	15	4824
94	svhn	[6, 8, 2, 0, 5]	25	15	5000
95	fashion-mnist	[Sandal, Sneaker, Pullover, Ankle boot, Bag]	25	15	5000
96	svhn	[4, 9, 2, 0, 1]	25	15	5000
97	cifar100	[beetle, motorcycle, skunk, bee, kangaroo]	25	15	500
98	cifar100	[rabbit, butterfly, rose, pear, mushroom]	25	15	500
99	svhn	[8, 7, 4, 2, 3]	25	15	5000
100	fashion-mnist	[Trouser, Sneaker, Bag, Tshirt/top, Pullover]	25	15	5000

Table 14: The 6 tasks created with Tiny-ImageNet. Classes that exist in different tasks are color-coded to highlight the class sharing.

	classes
task1	’n02124075’, ’n04067472’, ’n04540053’, ’n04099969’, ’n07749582’, ’n01641577’, ’n02802426’, ’n09246464’, ’n07920052’, ’n03970156’, ’n03891332’, ’n02106662’, ’n03201208’, ’n02279972’, ’n02132136’, ’n04146614’, ’n07873807’, ’n02364673’, ’n04507155’, ’n03854065’, ’n03838899’, ’n03733131’, ’n01443537’, ’n07875152’, ’n03544143’, ’n09428293’, ’n03085013’, ’n02437312’, ’n07614500’, ’n03804744’, ’n04265275’, ’n02963159’, ’n02486410’, ’n01944390’, ’n09256479’, ’n02058221’, ’n04275548’, ’n02321529’, ’n02769748’, ’n02099712’, ’n07695742’, ’n02056570’, ’n02281406’, ’n01774750’, ’n02509815’, ’n03983396’, ’n07753592’, ’n04254777’, ’n02233338’, ’n04008634’,
task2	’n09428293’, ’n03085013’, ’n02437312’, ’n07614500’, ’n03804744’, ’n04265275’, ’n02963159’, ’n02486410’, ’n01944390’, ’n09256479’, ’n02058221’, ’n04275548’, ’n02321529’, ’n02769748’, ’n02099712’, ’n07695742’, ’n02056570’, ’n02281406’, ’n01774750’, ’n02509815’, ’n03983396’, ’n07753592’, ’n04254777’, ’n02233338’, ’n04008634’, ’n02823428’, ’n02236044’, ’n03393912’, ’n07583066’, ’n04074963’, ’n01629819’, ’n09332890’, ’n02481823’, ’n03902125’, ’n03404251’, ’n09193705’, ’n03637318’, ’n04456115’, ’n02666196’, ’n03796401’, ’n02795169’, ’n02123045’, ’n01855672’, ’n01882714’, ’n02917067’, ’n02988304’, ’n04398044’, ’n02843684’, ’n02423022’, ’n02669723’
task3	’n02823428’, ’n02236044’, ’n03393912’, ’n07583066’, ’n04074963’, ’n01629819’, ’n09332890’, ’n02481823’, ’n03902125’, ’n03404251’, ’n09193705’, ’n03637318’, ’n04456115’, ’n02666196’, ’n03796401’, ’n02795169’, ’n02123045’, ’n01855672’, ’n01882714’, ’n02917067’, ’n02988304’, ’n04398044’, ’n02843684’, ’n02423022’, ’n02669723’, ’n04465501’, ’n02165456’, ’n03770439’, ’n02099601’, ’n04486054’, ’n02950826’, ’n03814639’, ’n04259630’, ’n03424325’, ’n02948072’, ’n03179701’, ’n03400231’, ’n02206856’, ’n03160309’, ’n01984695’, ’n03977966’, ’n03584254’, ’n04023962’, ’n02814860’, ’n01910747’, ’n04596742’, ’n03992509’, ’n04133789’, ’n03937543’, ’n02927161’
task4	’n04465501’, ’n02165456’, ’n03770439’, ’n02099601’, ’n04486054’, ’n02950826’, ’n03814639’, ’n04259630’, ’n03424325’, ’n02948072’, ’n03179701’, ’n03400231’, ’n02206856’, ’n03160309’, ’n01984695’, ’n03977966’, ’n03584254’, ’n04023962’, ’n02814860’, ’n01910747’, ’n04596742’, ’n03992509’, ’n04133789’, ’n03937543’, ’n02927161’, ’n01945685’, ’n02395406’, ’n02125311’, ’n03126707’, ’n04532106’, ’n02268443’, ’n02977058’, ’n07734744’, ’n03599486’, ’n04562935’, ’n03014705’, ’n04251144’, ’n04356056’, ’n02190166’, ’n03670208’, ’n02002724’, ’n02074367’, ’n04285008’, ’n04560804’, ’n04366367’, ’n02403003’, ’n07615774’, ’n04501370’, ’n03026506’, ’n02906734’

Table 15: (Continued) The 6 tasks created with Tiny-ImageNet. Classes that exist in different tasks are color-coded to highlight the class sharing.

	classes
task5	'n01945685', 'n02395406', 'n02125311', 'n03126707', 'n04532106',
	'n02268443', 'n02977058', 'n07734744', 'n03599486', 'n04562935',
	'n03014705', 'n04251144', 'n04356056', 'n02190166', 'n03670208',
	'n02002724', 'n02074367', 'n04285008', 'n04560804', 'n04366367',
	'n02403003', 'n07615774', 'n04501370', 'n03026506', 'n02906734',
	'n01770393', 'n04597913', 'n03930313', 'n04118538', 'n04179913',
	'n04311004', 'n02123394', 'n04070727', 'n02793495', 'n02730930',
	'n02094433', 'n04371430', 'n04328186', 'n03649909', 'n04417672',
	'n03388043', 'n01774384', 'n02837789', 'n07579787', 'n04399382',
	'n02791270', 'n03089624', 'n02814533', 'n04149813', 'n07747607'
task6	'n03355925', 'n01983481', 'n04487081', 'n03250847', 'n03255030',
	'n02892201', 'n02883205', 'n03100240', 'n02415577', 'n02480495',
	'n01698640', 'n01784675', 'n04376876', 'n03444034', 'n01917289',
	'n01950731', 'n03042490', 'n07711569', 'n04532670', 'n03763968',
	'n07768694', 'n02999410', 'n03617480', 'n06596364', 'n01768244',
	'n02410509', 'n03976657', 'n01742172', 'n03980874', 'n02808440',
	'n02226429', 'n02231487', 'n02085620', 'n01644900', 'n02129165',
	'n02699494', 'n03837869', 'n02815834', 'n07720875', 'n02788148',
	'n02909870', 'n03706229', 'n07871810', 'n03447447', 'n02113799',
	'n12267677', 'n03662601', 'n02841315', 'n07715103', 'n02504458'