Neural Lineage
- Supplementary Material -

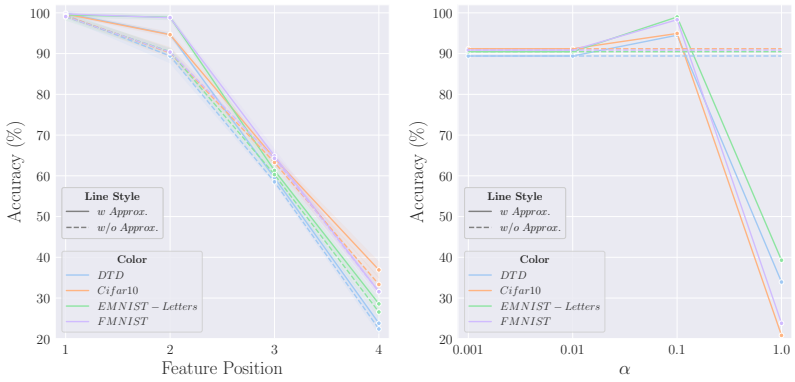
1. Further Experiments and Analyses

1.1. Learning-Free Methods

We note two crucial factors controlling the performance of the learning-free methods. The first is the position of the feature used, and the second is the hyper-parameter α. The following two figures present their influences.

For the feature position, we compare the performance of our method and that of the baseline when using ResNet18 on the DTD, Cifar10, EMNIST-Letters, and FMNIST. The similarity metric is chosen to be the l_1-norm, which is generally the best, as indicated by the results in Tab. 1 in the main text. We use the following four features to calculate the similarities, No. 1: the feature after the first convolutional layer, No. 2: the feature after the first ResNet block, No. 3: the feature after the second ResNet block, and No. 4: the feature after the third ResNet block. From No. 1 to No. 4, the feature used to evaluate the similarities gets deeper and deeper. For each feature position, we search for the best α. All other configurations are the same as the corresponding original experiments. As shown in Fig. 6a, although our method consistently outperforms the baseline, the accuracy of both our method and the baseline decreases with increasing feature depth. This trend may be attributed to the fact that as features become deeper, they encapsulate more high-level semantic information. The shallow features of different models might vary significantly, as there could be numerous ways to extract these low-level features while still achieving accurate classification. However, given that the models we selected all have quite high accuracy, their deeper features tend to align more closely with the classification objective. For instance, the mutual information between features and labels increases. Thus, this reduces the differences between the features of different models and leads to a decline in lineage detection accuracy, which results in an accuracy plot that is initially higher but decreases then.

For α, we again compare the performance of our method and that of the baseline when using ResNet18 on the DTD, Cifar10, EMNIST-Letters, and FMNIST. The similarity metric is still l_1-norm. We fix the feature position to be the feature after the first ResNet block. All other configurations are the same as the corresponding original experiments. The results are plotted in Fig. 6b. The variation in the value of α results in distinct experimental phenomena. With a small α, the learning-free method is nothing but the original similarity metric, which leads to no performance gain; with a large α, the finetuning approximation is inaccurate, which leads to a significant performance drop; with a proper α, the finetuning approximation is valid, our method outperforms the baseline.



(a) Feature Position (b) α Value

Figure 6. Comparison of the performance when feature position and α vary.

1.2. Lineage Detector

In practice, we may encounter scenarios where our parent model set is incomplete, and the child model is not fine-tuned from any model within the parent model set. To investigate this, we design a new experiment based on the "ResNet18+Cifar10" and "ResNet+SVHN" setups. During dataset construction, we excluded the "resnet18.tv_in1k" model from the parent model set but retained all its fine-tuned child models. This approach resulted in some child models having no known parent model. For child models other than those fine-tuned from "resnet18.tv_in1k", the ground truth labels remain unchanged, pointing to their original parents. However, the child models of "resnet18.tv_in1k" are assigned an additional category to indicate they have "no parent".

Consequently, we had six parent models. Using our framework, the lineage detector could output a six-dimensional vector $\boldsymbol{s} \in \mathcal{R}^6$ for each child model, representing the similarity scores with all six parent models. To enable the lineage detector to predict the "no parent", we concatenate a learnable scalar s' to the end of $\boldsymbol{s}$, forming a new vector $[\boldsymbol{s}_1; \cdots; \boldsymbol{s}_6; s']$, which

served as the final logits output of the lineage detector. During parent prediction, if $s' = \max\{\boldsymbol{s}_1, \cdots, \boldsymbol{s}_6, s'\}$, the model predicts that the child model's parent is not within the existing parent model set.

The rest of the experimental setup remains consistent with the original experiments. The results, presented in Tab. 4, show that while no-parent child models can sometimes disrupt lineage detection, such as in the Cifar10 setup, the lineage detector can also maintain a relatively high accuracy under the other setup. Moreover, our model could almost perfectly identify child models whose parent model does not belong to the current parent model set.

	Cifar10	SVHN
Overall Accuracy	82.14±1.58	99.26±0.12
Accuracy of the Children Without Real Parents	99.65±0.37	99.92±0.21

Table 4. The performance of Lineage Detector when some of the child models do not have real patients.

Benefiting from the structure of the transformer, the lineage detector allows for the use of multiple weights or features as inputs, or it can rely solely on either weights or features. We conducted experiments to analyze the impact of different combinations of weights and features on the performance of the lineage detector. These experiments were carried out under the "FC+FMNIST" and "FC+EMNIST-Letters" setups, with all other experimental conditions remaining consistent with the original settings, except for the variations in features and weights used. The results are summarized in Tab. 5. Comparing the rows of "Feature after the 1st ReLU Layer", "Weight of the 1st Linear Layer", and the corresponding experiments in the main text Tab. 1, it is evident that the lineage detector, using only features or weights, can surpass the performance of learning-free methods, which use both of them. However, comparing the last row with other rows, there is a noticeable performance gap between the lineage detector using either the weight or feature alone and the lineage detector using both of them. We hypothesize that the lineage detector, when provided with both, can learn an approximation of the fine-tuning process similar to Eq. (1) in the main text, thereby enhancing lineage detection effectiveness. Another observation is consistent with the phenomena in Fig. 6a: deeper features tend to decrease the effectiveness of lineage detection. Integrating features from multiple layers does not mitigate this issue.

	FMNIST	EMNIST
Feature after the 1st ReLU Layer	54.58±3.31	50.01±7.37
Feature after the 2nd ReLU Layer	9.02±2.86	14.86±3.93
Feature after the 3rd ReLU Layer	9.08±3.08	12.75±2.06
Feature after the 1st ReLU Layer + Feature after the 2nd ReLU Layer	40.92±2.28	45.56±5.09
Weight of the 1st Linear Layer	57.34±4.97	62.94±5.81
Weight of the 2nd Linear Layer	76.98±4.09	91.24±1.99
Weight of the 3rd Linear Layer	89.11±2.51	89.23±2.02
Weight of the 1st Linear Layer + Weight of the 2nd Linear Layer	66.32±6.01	87.91±4.03
Feature after the 1st ReLU Layer + Weight of the 1st Linear Layer	**97.86±1.19**	**93.61±2.37**

Table 5. The performance of Lineage Detector when different weight and feature combination is used.

1.3. Average Performance Across Datasets

To facilitate the comparison of the performance of various methods, we calculate the average performance of each method across classification datasets. The results are shown in Tab. 6. The learning-based method maintaining the highest accuracy.

	Without Approximation				With Approximation				Lineage Detector
	l_1	l_2	CKA	DC	l_1	l_2	CKA	DC	
FCNet	63.41±1.27	65.03±1.41	26.22±1.18	26.65±1.19	64.95±1.49	<u>66.66±1.44</u>	32.27±1.25	32.17±1.41	**98.29±0.90**
ResNet18	92.72±0.91	91.67±0.83	64.54±1.82	56.23±1.95	<u>97.31±0.60</u>	92.52±0.92	66.86±2.12	56.99±1.94	**99.30±0.23**

Table 6. The average performance of various methods across different datasets in Tab. 1 of the main text. The best (the second-best) ones are marked in **bold** (<u>underlined</u>).

Among the learning-free methods, l_1 shows superiority for ResNet, while l_2 excels for Fully Connected Network. Norm distance consistently outperforms advanced representation similarity.

2. Implementation Details

Experiments are conducted using Nvidia RTX 3090 if no special specification.

2.1. Classification Model Experiments in Table 1

2.1.1 Parent and Child Models

We employ two families of classification models: one comprising three-layer fully connected networks only with ReLU as the nonlinear activation, and the other is the ResNet18.

For experiments using a fully connected network on FMNIST and EMNIST-Letters, the input and output dimensions of the three-layer fully connected network models are adjusted according to the dataset, with hidden feature sizes being fixed to 1024, 256, and 128. We initially train 20 parent models on MNIST, each with different training configurations. Their batch sizes are one of $\{64, 256, 1024\}$, and their learning rates are one of $\{0.1, 0.01, 0.001\}$. We use four different random seeds. The training epochs are set to 50. We use the Adam optimizer to train. From the 36 trained models, we select the top 20 with the highest test accuracy as our parent models for experiments. Starting from these 20 parent models, we fine-tune them on FMNIST and EMNIST-Letters datasets. The fine-tuning learning rates are one of $\{0.01, 0.001, 0.0001\}$, and batch sizes are one of $\{1024, 256\}$. We again use four different random seeds. The tuning epochs are set to 36. We use the Adam optimizer. Eventually, on each dataset, we obtain 480 models and select those with test accuracy above 80% as our child models for subsequent experiments. This results in 228 child models each for FMNIST and EMNIST-Letters.

For experiments using a fully connected network on Cifar10, SVHN, and variants of Cifar10, the input and output dimensions of the three-layer fully connected network models are adjusted according to the dataset, with hidden feature sizes being fixed to 1024, 256, and 128. We initially train 12 parent models on Cifar100, each with different training configurations. Their batch sizes are either 256 or 1024, and their learning rates are either 0.01 or 0.001. We use three different random seeds. The training epochs are set to 50. We use the Adam optimizer to train. We select all of them as the parent models, as their testing accuracy are all above 80%. Starting from these 12 parent models, we fine-tune on Cifar10, SVHN, and Cifar10 variants. The fine-tuning hyperparameters are similar to the previous setup, with learning rates of 0.01, 0.001, or 0.0001, and batch sizes of 1024 or 256, using four different random seeds. The training epochs are set to 20, using the Adam optimizer. Eventually, on each dataset, we obtain 288 models, selecting those with test accuracy above 80% as our child models for subsequent experiments. This resulted in 254, 264, 132, 264, and 260 child models for Cifar10, SVHN, Cifar10-5shot, Cifar10-50shot, and Cifar10-imbalanced datasets, respectively.

For the experiments using a ResNet18 network, we use 7 ResNet18 networks in the timm [4] package as the parent models. All of the networks are trained on the ImageNet dataset. The networks we used are "resnet18.a1_in1k", "resnet18.a2_in1k", "resnet18.a3_in1k", "resnet18.gluon_in1k", "resnet18.fb_ssl_yfcc100m_ft_in1k", "resnet18.fb_swsl_ig1b_ft_in1k" and "resnet18.tv_in1k". If the finetuning dataset is FMNIST or EMNIST-Letters, the kernel of the first convolutional layer is replaced by its average along the channel dimension to support grayscale input. We fine-tune these 7 models on the datasets listed in Tab. 1. The fine-tuning hyperparameters are similar to the previous setups, with learning rates of 0.01, 0.001, or 0.0001, and batch sizes of 1024 or 256, using four different random seeds. The training epochs are set to 20. The optimizer is Adam. The head of the network is modified according to the number of categories of the target dataset and reinitialized. Eventually, on each dataset, we obtain 168 models and select those with test accuracy above 80% as our child models for subsequent experiments. For each of FMNIST, EMNIST-Letters, Cifar10, SVHN, DTD, Pets, and Cifar10-imbalanced datasets, we obtain 168 child models. For Cifar10-5shot and Cifar10-50shot, we obtain 90 and 94 child models respectively.

2.1.2 Our Methods and Baselines

When the fully connected network is used, both our learning-free methods and the baselines use the output feature of the first ReLU layer to calculate the similarity. When ResNet18 is used, both our learning-free methods and the baselines use the output feature of the first ResNet block to calculate the similarity. Though our learning-based lineage detector is able to use only the feature as input, for a fair comparison, we use both the weight and the feature as input in this set of experiments. When the fully connected network is used, the weight of the first linear layer and the output feature of the first ReLU layer are taken as the input of the lineage detector; when ResNet18 is used, the weight of the last linear layer in the first ResNet

block and the output feature of the first ResNet block are taken as the input of the lineage detector. For the learning-free method, the hyper-parameter α is searched within $\{0.001, 0.01, 0.1\}$. For all the methods, the number of samples to evaluate the similarity is set to 512. The parameter p in p-norm is always set to be 4. The parameter t in log-sum-exp is always set to 0.01.

2.2. Classification Model Experiments in Table 2

In this experiment, we evaluated our method's capability for cross-generational detection across four generations of models. For the first vs. second-generation experiment, the parent and child models are those used in the ResNet+EMNIST-Letters experiment in Tab. 1. We refer to the first-generation models as root models, with all subsequent models being their descendants. For each root model, we select the second-generation child model with the highest test accuracy as the parent model for the second vs. third-generation experiment. These models are fine-tuned on the FMNIST dataset to obtain the third-generation models, which served as child models in the second vs. third-generation experiment. Similarly, for each root model, we select the third-generation descendant with the highest test accuracy as the parent model for the third vs. fourth-generation experiment. These models are fine-tuned on the EMNIST-Balanced dataset to obtain the fourth-generation models, which are the child models in the third vs. fourth-generation experiment. The fine-tuning hyperparameters are consistent with previous setups, including learning rates of 0.01, 0.001, or 0.0001, batch sizes of 1024 or 256, and four different random seeds. The training epochs are set to 20, using the Adam optimizer, and the network head is modified and reinitialized according to the number of categories in the target dataset. Ultimately, we obtained 168 models for each dataset, selecting only those with test accuracy above 80% for further experiments.

Through this approach, we constructed a model family comprising 7 first-generation models; 168 second-generation models, all of which are child models in the first vs. second-generation experiment, with 7 serving as parent models in the second vs. third-generation experiment; 127 third-generation models, all child models in the second vs. third-generation experiment, with 7 serving as parent models in the third vs. fourth generation experiment; and 168 fourth-generation models, all child models in the third vs. fourth generation experiment. For the first vs. third and first vs. fourth-generation experiments, we used the 7 root models as parent models and the third and fourth-generation models as child models. For the second vs. fourth-generation experiment, the 7 parent models from the second vs. third-generation experiment were used as parent models, with the fourth-generation models as child models.

The implementation of our methods and the baselines is consistent with the settings used in the ResNet+EMNIST-Letters experiment in Tab. 1.

2.3. Detection and Segmentation Model Experiments

2.3.1 Parent and Child Models

For the detection and segmentation experiments, we utilized the official DETR+ResNet50 model released for detection and segmentation, respectively. We first created two subsets of the PASCAL dataset: SubSet 1, containing all categories except animals, and SubSet 2, comprising all animal categories. For detection, we fine-tune the official "DETR_R50_detection" model on SubSet 1. The fine-tuning learning rate is either 0.005, 0.001, 0.0005, or 0.0001, the batch size is 16, and the epoch is 25. We use two different random seeds. All other experimental settings follow the default training configurations of DETR. This process yielded 8 models. We use all of them as parent models. These parent models are further fine-tuned on SubSet 2. The fine-tuning learning rate is either 0.005, 0.001, 0.0005, or 0.0001, the batch size is 16, and the epoch is 25. We use four different random seeds. This resulted in 128 models, from which we select those with training accuracy above 80%, resulting in 56 child models.

For the segmentation experiment, we fine-tune the official "DETR_R50_segmentation" model on SubSet 1. The fine-tuning learning rate is either 0.005, 0.001, 0.0005, or 0.0001, the batch size is 16, and the epoch is 25. We use two different random seeds. All other experimental settings follow the default training configurations of DETR. This process yielded 8 models. We use all of them as parent models. These parent models are further fine-tuned on SubSet 2. The fine-tuning learning rate is either 0.005, 0.001, 0.0005, or 0.0001, the batch size is 16, and the epoch is 25. We use four different random seeds. This resulted in 128 models, from which we select those with training accuracy above 80%, resulting in 49 child models.

2.3.2 Our Methods and Baselines

For both the detection and segmentation, both our learning-free methods and the baselines use the output feature of the first transformer block in the encoder of DETR. For both the detection and segmentation, the weight of the last linear layer in the first transformer block in the encoder of DETR and the output feature of the first transformer block in the encoder of DETR

are taken as the input of the lineage detector. For the learning-free method, the hyper-parameter α is searched within $\{0.001, 0.01, 0.1\}$. For all the methods, the number of samples to evaluate the similarity is set to 16. The parameter p in p-norm is always set to be 4. The parameter t in log-sum-exp is always set to 0.01.

2.4. Continual Learning and Knowledge Distillation Experiments

We investigate the impact of two types of loss functions with additional regularization terms on lineage detection: Elastic Weight Consolidation (EWC) from Continual Learning and Kullback-Leibler Divergence (KLD) from Knowledge Distillation.

The EWC experiment involves incorporating EWC loss into the original FC+FMNIST setup. The implementation of EWC loss adheres to the methodology outlined in the original paper. [3] The weight of the EWC loss is set at 10, 20, or 100. The number of random seeds is reduced to two. The selection of parent models, other settings for tuning child models, and the method for filtering child models remain consistent with the FC+FMNIST setup. Ultimately we obtain 227 child models.

In the KLD experiment, an additional teacher model and KL divergence loss are introduced alongside the original ResNet18+FMNIST setup. This experiment utilizes "resnet18.a1_in1k", "resnet18.a2_in1k", "resnet18.a3_in1k", "resnet18.gluon_in1k", "resnet18.fb_swsl_ig1b_ft_in1k" and "resnet18.tv_in1k" as parent models. The child model with the highest test accuracy fine-tuned from the "resnet18.fb_ssl_yfcc100m_ft_in1k" model on FMNIST is used as the teacher model. During the tuning of child models from parent models, a KLD term is added, requiring the final output of the child models to be similar in KLD to the final output of the teacher model. This approach is in line with standard response-based knowledge distillation practices. [1] The weight of the KLD loss is set to 1, with the temperature parameter in the KLD loss adjusted to 1.0, 2.0, or 5.0. The number of random seeds is again reduced to two. Other settings for training child models and the method for filtering child models are consistent with the ResNet18+FMNIST setup, resulting in 168 child models.

In the continual learning experiment, the implementation of our methods and the baselines is consistent with the settings used in the FC+FMNIST experiment in Tab. 1. In the knowledge distillation experiment, the implementation of our methods and the baselines is consistent with the settings used in the ResNet18+FMNIST experiment in Tab. 1.

2.5. Lineage Detector

2.5.1 Dataset for Lineage Detector

The prerequisites of training and utilizing the lineage detector involve constructing an appropriate dataset. Assuming no constraints on sample size, a dataset suitable for training and testing the lineage detector can be constructed from each of our "neural network architecture + finetuning dataset" settings. Taking the "FC+FMNIST" setup as an example, we illustrate how to construct a dataset for the lineage detector, applicable similarly to other settings. We assume that the input weights for the lineage detector are all parameters of the neural network except for the head, and the input feature is the backbone output, *i.e.*, the input feature of the neural network head. If one opts not to use weights or features as inputs, they can be set as zero matrices, and the corresponding encoder in the lineage detector can be omitted.

Let $f_p : \mathcal{X} \to \mathcal{Y}$ and $f_c : \mathcal{X} \to \mathcal{Y}$ denote the parent neural network and the child neural network parameterized by $\boldsymbol{\theta}_p$ and $\boldsymbol{\theta}_c$, respectively. Suppose there are M parent models $\{f_p^{(m)}\}_{m=1}^{M}$ and M_c child models $\{f_c^{(m)}\}_{m=1}^{M_c}$. Suppose there are N FMNIST samples $\{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})\}_{i=1}^{N}$. Let $\boldsymbol{X}$ denote the images in batch format. Let $\tilde{\boldsymbol{\theta}}$ denote the parameters except the parameters in the head. Let $\tilde{f}$ denote the neural network without the head.

First, we collect the parameters $\tilde{\boldsymbol{\theta}}$ of all parent and child models, $\{\tilde{\boldsymbol{\theta}}_p^{(m)}\}_{m=1}^{M}$ and $\{\tilde{\boldsymbol{\theta}}_c^{(m)}\}_{m=1}^{M_c}$. Second, we evaluate the backbone output of parent and child models, $\{\tilde{f}_p^{(m)}(\boldsymbol{X})\}_{m=1}^{M}$ and $\{\tilde{f}_c^{(m)}(\boldsymbol{X})\}_{m=1}^{M_c}$. For each of the $\tilde{\boldsymbol{\theta}}_p^{(m)}$, $\tilde{\boldsymbol{\theta}}_c^{(m)}$, $\tilde{f}_p^{(m)}(\boldsymbol{X})$, and $\tilde{f}_c^{(m)}(\boldsymbol{X})$ obtained, we vectorize it and reshape them into a matrix, such that, $\tilde{\boldsymbol{\theta}}_p^{(m)} \in \mathcal{R}^{H_\theta \times W_\theta}$, $\tilde{\boldsymbol{\theta}}_c^{(m)} \in \mathcal{R}^{H_\theta \times W_\theta}$, $\tilde{f}_p^{(m)}(\boldsymbol{X}) \in \mathcal{R}^{H_F \times W_F}$, and $\tilde{f}_c^{(m)}(\boldsymbol{X}) \in \mathcal{R}^{H_F \times W_F}$, where W_θ, H_θ, W_F, and H_F indicate the shape of the matrix.

With $\{\tilde{\boldsymbol{\theta}}_p^{(m)}\}_{m=1}^{M}$, $\{\tilde{f}_p^{(m)}(\boldsymbol{X})\}_{m=1}^{M}$, $\tilde{\boldsymbol{\theta}}_c^{(n)}$ and $\tilde{f}_c^{(n)}(\boldsymbol{X})$ we can generate the probabilities of the parent models for the child model n. Thus, in the dataset for lineage detector, each sample contains five parts, the parent weights $\{\tilde{\boldsymbol{\theta}}_p^{(m)}\}_{m=1}^{M}$, the parent features $\{\tilde{f}_p^{(m)}(\boldsymbol{X})\}_{m=1}^{M}$, the weight of a specific child model $\tilde{\boldsymbol{\theta}}_c$, the feature of a specific child model $\tilde{f}_c(\boldsymbol{X})$ and the index of the real parent of this child model as the ground truth label. The number of samples in a dataset for lineage detector is the number of child models.

Based on this dataset, once the partition of training and test sets is completed, training and testing can proceed.

2.5.2 Model and Training Configuration

Both the weight encoder and the feature encoder are two-layer convolutional neural networks. The number of the input channels of the first layer is 2. The number of the output channels of the first layer is set to 3. The number of the output channels of the second layer is set to 32. The kernel size of the first layer is 1. The kernel size of the second layer is 3. The transformer detector encompasses one transformer layer with all default settings in PyTorch.

The training learning rate is 0.01, the batch size is 1, and the epoch is 100 for all experiments. The Adam optimizer is used.

2.6. Sub-Figure 1b

This visualization represents the experimental results of the classification models discussed in the main text. The figure encompasses four generations of models. We select "resnet18.a1_in1k", "resnet18.a2_in1k", "resnet18.a3_in1k", "resnet18.gluon_in1k" and "resnet18.tv_in1k" as the root models. These five models are all used in our experiments of the ResNet in the main text. We mark these five models as the models of the first generation. Models fine-tuned on EMNIST-Letters from these root models constitute the second generation. The third generation comprises models fine-tuned on FMINST from the second generation, and the fourth generation includes models fine-tuned on EMNIST-Balanced from the third generation. The total number of models is 226. Due to space constraints, not all models used in our experiments are depicted; only the five root models and their descendants are included in this visualization.

The diagram features three concentric rings. The innermost ring consists of multiple small arches, each representing a neural network. The arches vary in width, with the width being proportional to the number of descendants of that model. The middle ring has 20 arches, divided into five color groups, each color representing a different root model. Models covered by the arches with the same color share the same root model, indicating a common ancestor. The outer ring is segmented into four colors, corresponding to the generation of the models; models covered by the arches with the same color belong to the same generation.

At the center of the diagram, arcs with three different colors represent our predictions of the model lineage, with each color indicating a different type of ancestor, such as parent, grandparent, or great-grandparent. Our lineage detector accurately predicts the lineage relationships of all model pairs. Some arcs in the diagram have increased transparency solely for visual clarity and do not carry additional meaning.

2.7. Sub-Figure 1c

We utilized nine ViT-Base models from the timm library to construct our hybrid model. The name of these nine models in the timm are: "vit_base_patch16_224.augreg_in21k_ft_in1k", "vit_base_patch32_224.sam_in1k", "vit_base_r50_s16_224.orig_in21k", "vit_base_patch8_224.augreg2_in21k_ft_in1k", "vit_base_patch16_224.mae", "vit_base_patch32_clip_224.openai_ft_in1k", and "vit_base_patch32_clip_224.laion2b_ft_in12k_in1k". The first 6 models are released by Google, the 7th model is released by Meta, and the last 2 models are released by OpenAI. We construct the hybrid model by the following steps. First, starting with the model "vit_base_patch16_224.augreg_in21k_ft_in1k", we replace its third transformer layer with the third transformer layer from the model "vit_base_patch32_224.sam_in1k". Now we have a hybrid model. Continuously, we replace the fourth transformer layer of the hybrid model with the fourth transformer layer from the model "vit_base_r50_s16_224.orig_in21k". We keep doing this until the third transformer layer to the tenth transformer layer are all replaced by the corresponding layers in another model. The final hybrid model is shown in the SubFig. 1c. We then reinitialized the linear head of this hybrid model and fine-tuned it on the CIFAR-10 dataset. The fine-tuning process used a learning rate of 1e-4, a batch size of 128, and epochs 30, with the Adam optimizer. Prior to fine-tuning, the hybrid model had an accuracy of 0%, which increased to 100% on the training set and 45% on the test set after fine-tuning.

Next, we tend to predict the origin of the transformer layers in the hybrid model. For the lineage detection of the i-th transformer layer, the sub-network of the hybrid model up to the i-th transformer layer is regarded as the child model. The candidate parent models are constructed as follows. We concatenate the sub-network of the hybrid model before the i-th transformer layer with the i-th transformer layer of each of the nine parent models, resulting in nine new networks. These networks had the same input as the original ViT-Base model and their outputs have the sample shape as output of the i-th transformer layer of the original ViT model. We fed the same data into these nine networks to obtain their outputs, which served as the parent feature inputs for our lineage detector. The features of the i-th transformer layer of the hybrid model, under the same input, are used as the child feature inputs for the lineage detector. In this task, we only use features without any network weights as the input of the lineage detector. For efficiency, we used a sample size of 128 and only the first 64

tokens of the feature. Because there is not enough data in this ViT setting to train the lineage detector. We turn to another setting we used. We use the parent and child models in the "FC + FMNIST" setting. We use the output feature of the first ReLU of the fully connected network as the training data to train the lineage detector.

There is another detail worth discussing. Because the probability distribution of the softmax function can be modified by scaling the input of the softmax function. With a large enough scaling parameter, the probability distribution can be made close to a one-hot vector, which might appear to offer high certainty in predictions and indicate the superior performance of the model. However, we believe this to be misleading and not truly reflective of the model's predictive output. Therefore, before inputting into the softmax, we normalized all values by correcting their mean to be zero and dividing them by the number of parent models 9. Such adjustment does not change the prediction and the sorting results of the probability output of the softmax function, only influences the value of the probability.

2.8. Figure 2

The experiment is conducted on the Cifar10 dataset. The number of samples used to calculate the similarity is 10. To show the influence of the number of parameters, we fix the output feature size to be 5 and compare ResNet18, ResNet34, and ResNet50, whose number of parameters are roughly 12M, 22M, and 25M respectively. To show the influence of output feature size, we fix the network to be ResNet18 and change its output dimension to 5, 10 or 15 by modifying its final fc layer. To avoid mutual interference between experiments, the experiments are conducted one by one. The server used to conduct the experiments has one RTX 2080 Ti and 16-core Intel(R) Xeon(R) Platinum 8255C CPU.

2.9. Sub-Figure 3a

In this experiment, we calculate the similarity between one parent model and its 20 child models using the step-by-step method and the proposed approximation. The network architecture is a three-layer fully connected network only with ReLU as the non-linear activation. The parent model is trained on MNIST dataset with a learning rate of 0.01 and a batch size of 64. The child models are fine-tuned on the FMNIST dataset with a learning rate of 0.001. The finetuning batch size is one in $\{1024, 256\}$. The finetuning iteration is one in $\{100, \cdots, 1000\}$. The number of samples used to calculate the similarity is 10. The number of output features is set to 10.

2.10. Sub-Figure 3b

In this experiment, we compare the distribution of the change of the similarity after the finetuning process is taken into consideration. We use a three-layer fully connected network only with ReLU as the non-linear activation as the real parent model. This model is trained on the MNIST dataset with a learning rate of 0.01 and a batch size of 64. All the child models are fine-tuned from this model. In total, we use 24 child models. They are fine-tuned on the FMNIST dataset. The finetuning batch size is one in $\{1024, 256\}$. The finetuning learning rate is one in $\{0.1,0.01,0.001\}$. The finetuning iteration in one in $\{100, 200\}$. We also use two different random seeds. We train another model on MNIST dataset with a learning rate of 0.01 and a batch size of 1024 as the fake parent model. The number of samples used to calculate the similarity is 5000. The number of output features is set to 10. For each parent-child model pair, no matter whether the parent is fake or real, we first calculate the similarity without the approximation of the finetuning process, then calculate the similarity with the approximation of the finetuning process using our proposed method, and finally, we evaluate the change in similarity by subtracting the former value form the latter one.

3. Derivations

This section includes the derivations for the Proposition 1 in the main text. Before the derivations, we introduce some notations to simplify the formulation and ease the reading. Let $f(\mathcal{X}) \in \mathbb{R}^{N\times K}$ denote the output of the neural network in the matrix form taking the entire dataset as the input with each row corresponding to output for one data sample. Specifically,

$$
\begin{aligned}
&\boldsymbol{X} \triangleq f_p(\mathcal{X}) \triangleq [f_p(\boldsymbol{x}^{(1)}), \cdots, f_p(\boldsymbol{x}^{(N)})]^T,\\
&\bar{\boldsymbol{X}} \triangleq \bar{f}_p(\mathcal{X}) \triangleq [\bar{f}_p(\boldsymbol{x}^{(1)}), \cdots, \bar{f}_p(\boldsymbol{x}^{(N)})]^T,\\
&\boldsymbol{Y} \triangleq f_c(\mathcal{X}) \triangleq [f_c(\boldsymbol{x}^{(1)}), \cdots, f_c(\boldsymbol{x}^{(N)})]^T,\\
&\boldsymbol{G} \triangleq \nabla_{\boldsymbol{\theta}_p} f_p(\mathcal{X})(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) \triangleq [\nabla_{\boldsymbol{\theta}_p} f_p(\boldsymbol{x}^{(1)})(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p), \cdots, \nabla_{\boldsymbol{\theta}_p} f_p(\boldsymbol{x}^{(N)})(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)]^T
\end{aligned}
$$

where the single letter $\boldsymbol{X}$, $\bar{\boldsymbol{X}}$, $\boldsymbol{Y}$, and $\boldsymbol{G}$ is to further simplify the notation. For a matrix $\boldsymbol{A} \in \mathbb{R}^{a\times b}$, let $\boldsymbol{A}_{:k} \in \mathbb{R}^{a\times 1}$, $\boldsymbol{A}_{i:} \in \mathbb{R}^{1\times b}$, and $\boldsymbol{A}_{ik} \in \mathbb{R}$ denote the k-th column, the i-th row, and the element at i-th row and j-th column. Let δ denote the high-order terms, which are omitted. Let $\mathbf{1}$ denote the all-one column vector with proper dimension. Let $\boldsymbol{I}$ denote the identity matrix with proper dimension. Let $\boldsymbol{H} \triangleq \boldsymbol{I} - \frac{1}{N}\mathbf{1}\mathbf{1}^T$ denote the centering matrix,

3.1. Linear Approximation of L_1 Norm

$$
\begin{aligned}
s(\bar{f}_p, f_c) &= -\frac{1}{NK}\sum_{i=1}^{N}||\bar{\boldsymbol{X}}_{i:} - \boldsymbol{Y}_{i:}||_1 && (1)\\
&= -\frac{1}{NK}\sum_{i=1}^{N}||\boldsymbol{X}_{i:} + \boldsymbol{G}_{i:} - \boldsymbol{Y}_{i:}||_1 && (2)\\
&= -\frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}|\boldsymbol{X}_{ik} + \boldsymbol{G}_{ik} - \boldsymbol{Y}_{ik}| && (3)\\
&\approx -\frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}[|\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik}| + \text{sign}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\boldsymbol{G}_{ik}] + \delta && (4)\\
&= -\frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}|\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik}| - \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}\text{sign}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\boldsymbol{G}_{ik} + \delta && (5)\\
&= s(f_p, f_c) - \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}\text{sign}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\boldsymbol{G}_{ik} + \delta && (6)\\
&= s(f_p, f_c) - \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}\text{sign}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\nabla_{\boldsymbol{\theta}_p}\boldsymbol{X}_{ik}(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) + \delta && (7)\\
&= s(f_p, f_c) - \frac{1}{NK}\nabla_{\boldsymbol{\theta}_p}[\sum_{i=1}^{N}\sum_{k=1}^{K}\text{sign}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\boldsymbol{X}_{ik}](\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) + \delta && (8)\\
&= s(f_p, f_c) - \frac{1}{NK}\nabla_{\boldsymbol{\theta}_p}\langle \text{sg}[\text{sign}(\boldsymbol{X} - \boldsymbol{Y})], \boldsymbol{X}\rangle(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) + \delta && (9)
\end{aligned}
$$

3.2. Linear Approximation of L_2 Norm

$$
\begin{aligned}
s(\bar{f}_p, f_c) &= -\frac{1}{NK}\sum_{i=1}^{N}||\bar{\boldsymbol{X}}_{i:} - \boldsymbol{Y}_{i:}||_2^2 && (10)\\
&= -\frac{1}{NK}\sum_{i=1}^{N}||\boldsymbol{X}_{i:} + \boldsymbol{G}_{i:} - \boldsymbol{Y}_{i:}||_2^2 && (11)\\
&= -\frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}(\boldsymbol{X}_{ik} + \boldsymbol{G}_{ik} - \boldsymbol{Y}_{ik})^2 && (12)\\
&\approx -\frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}[(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})^2 + 2(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\boldsymbol{G}_{ik}] + \delta && (13)\\
&= -\frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})^2 - \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}2(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\boldsymbol{G}_{ik} + \delta && (14)\\
&= s(f_p, f_c) - \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}2(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\boldsymbol{G}_{ik} + \delta && (15)
\end{aligned}
$$

$$= s(f_p, f_c) - \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K} 2(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\nabla_{\boldsymbol{\theta}_p}\boldsymbol{X}_{ik}(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) + \delta \tag{16}$$

$$= s(f_p, f_c) - \frac{2}{NK}\nabla_{\boldsymbol{\theta}_p}[\sum_{i=1}^{N}\sum_{k=1}^{K}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\boldsymbol{X}_{ik}](\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) + \delta \tag{17}$$

$$= s(f_p, f_c) - \frac{2}{NK}\nabla_{\boldsymbol{\theta}_p}\langle \text{sg}(\boldsymbol{X} - \boldsymbol{Y}), \boldsymbol{X}\rangle(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) + \delta \tag{18}$$

3.3. Linear Approximation of L_p Norm

$$s(\bar{f}_p, f_c) = -\frac{1}{NK}\sum_{i=1}^{N}||\bar{\boldsymbol{X}}_{i:} - \boldsymbol{Y}_{i:}||_p^p \tag{19}$$

$$= -\frac{1}{NK}\sum_{i=1}^{N}||\boldsymbol{X}_{i:} + \boldsymbol{G}_{i:} - \boldsymbol{Y}_{i:}||_p^p \tag{20}$$

$$= -\frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}|\boldsymbol{X}_{ik} + \boldsymbol{G}_{ik} - \boldsymbol{Y}_{ik}|^p \tag{21}$$

$$\approx -\frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}[|\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik}|^p + p|\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik}|^{p-1}\text{sign}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\boldsymbol{G}_{ik}] + \delta \tag{22}$$

$$= -\frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}|\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik}|^p - \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}p|\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik}|^{p-1}\text{sign}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\boldsymbol{G}_{ik} + \delta \tag{23}$$

$$= s(f_p, f_c) - \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}p|\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik}|^{p-1}\text{sign}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\boldsymbol{G}_{ik} + \delta \tag{24}$$

$$= s(f_p, f_c) - \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}p|\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik}|^{p-1}\text{sign}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\nabla_{\boldsymbol{\theta}_p}\boldsymbol{X}_{ik}(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) + \delta \tag{25}$$

$$= s(f_p, f_c) - \frac{p}{NK}\nabla_{\boldsymbol{\theta}_p}[\sum_{i=1}^{N}\sum_{k=1}^{K}|\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik}|^{p-1}\text{sign}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})\boldsymbol{X}_{ik}](\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) + \delta \tag{26}$$

$$= s(f_p, f_c) - \frac{p}{NK}\nabla_{\boldsymbol{\theta}_p}\langle \text{sg}[|\boldsymbol{X} - \boldsymbol{Y}|^{p-1}\text{sign}(\boldsymbol{X} - \boldsymbol{Y})], \boldsymbol{X}\rangle(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) + \delta \tag{27}$$

3.4. Linear Approximation of $log - sum - exp$ Function

$$s(\bar{f}_p, f_c) = -\frac{1}{NKt}\sum_{i=1}^{N}\log\sum_{k=1}^{K}e^{t|\bar{\boldsymbol{X}}_{ik} - \boldsymbol{Y}_{ik}|} \tag{28}$$

$$= -\frac{1}{NKt}\sum_{i=1}^{N}\log\sum_{k=1}^{K}e^{t|\boldsymbol{X}_{ik} + \boldsymbol{G}_{ik} - \boldsymbol{Y}_{ik}|} \tag{29}$$

$$\approx -\frac{1}{NKt}\sum_{i=1}^{N}\log\sum_{k=1}^{K}e^{t|\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik}|} - \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}\frac{\boldsymbol{G}_{ik}e^{t|\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik}|}\text{sign}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})}{\sum_{j=1}^{K}e^{t|\boldsymbol{X}_{ij} - \boldsymbol{Y}_{ij}|}} + \delta \tag{30}$$

$$= s(f_p, f_c) - \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}\frac{\boldsymbol{G}_{ik}e^{t|\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik}|}\text{sign}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})}{\sum_{j=1}^{K}e^{t|\boldsymbol{X}_{ij} - \boldsymbol{Y}_{ij}|}} + \delta \tag{31}$$

$$= s(f_p, f_c) - \frac{1}{NK}\sum_{i=1}^{N}\sum_{k=1}^{K}[\frac{e^{t|\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik}|}\text{sign}(\boldsymbol{X}_{ik} - \boldsymbol{Y}_{ik})}{\sum_{j=1}^{K}e^{t|\boldsymbol{X}_{ij} - \boldsymbol{Y}_{ij}|}}\nabla_{\boldsymbol{\theta}_p}\boldsymbol{X}_{ik}(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)] + \delta \tag{32}$$

$$= s(f_p, f_c) - \frac{1}{NK}\nabla_{\boldsymbol{\theta}_p}\langle \text{sg}[\text{softmax}(t|\boldsymbol{X}-\boldsymbol{Y}|)\text{sign}(\boldsymbol{X}-\boldsymbol{Y})], \boldsymbol{X}\rangle(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) + \delta \tag{33}$$

3.5. Linear Approximation of Centered Kernel Alignment

The calculation of the squared CKA with the linear kernel can be written as

$$s(\bar{f}_p, f_c) = \frac{\langle \boldsymbol{H}\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^T\boldsymbol{H}, \boldsymbol{H}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H}\rangle^2}{\langle \boldsymbol{H}\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^T\boldsymbol{H}, \boldsymbol{H}\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^T\boldsymbol{H}\rangle\langle \boldsymbol{H}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H}, \boldsymbol{H}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H}\rangle}, \tag{34}$$

which can be evaluated step by step after substituting the definition of $\bar{\boldsymbol{X}}$ into it,

$$\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^T = \boldsymbol{X}\boldsymbol{X}^T + \boldsymbol{G}\boldsymbol{X}^T + \boldsymbol{X}\boldsymbol{G}^T + \delta, \tag{35}$$

$$\boldsymbol{H}\bar{\boldsymbol{X}}\bar{\boldsymbol{X}}^T\boldsymbol{H} = \boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H} + \boldsymbol{H}(\boldsymbol{G}\boldsymbol{X}^T + \boldsymbol{X}\boldsymbol{G}^T)\boldsymbol{H} + \delta, \tag{36}$$

$$\langle \boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H}, \boldsymbol{H}(\boldsymbol{G}\boldsymbol{X}^T + \boldsymbol{X}\boldsymbol{G}^T)\boldsymbol{H}\rangle = 2\sum_{i=1}^{N}[\nabla_{\boldsymbol{\theta}_p} f_p(\boldsymbol{x}^{(i)})(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)]^T(\boldsymbol{X} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\boldsymbol{X})^T(\boldsymbol{H}_{i:}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H} - \frac{1}{N}\boldsymbol{1}^T\boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H})^T \tag{37}$$

$$= \nabla_{\boldsymbol{\theta}_p}\Big[2\sum_{i=1}^{N}\big[\text{sg}[(\boldsymbol{H}_{i:}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H} - \frac{1}{N}\boldsymbol{1}^T\boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H})(\boldsymbol{X} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\boldsymbol{X})]f_p(\boldsymbol{x}^{(i)})\big]\Big](\boldsymbol{\theta}_c - \boldsymbol{\theta}_p), \tag{38}$$

$$\langle \boldsymbol{H}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H}, \boldsymbol{H}(\boldsymbol{G}\boldsymbol{X}^T + \boldsymbol{X}\boldsymbol{G}^T)\boldsymbol{H}\rangle = 2\sum_{i=1}^{N}[\nabla_{\boldsymbol{\theta}_p} f_p(\boldsymbol{x}^{(i)})(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)]^T(\boldsymbol{X} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\boldsymbol{X})^T(\boldsymbol{H}_{i:}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H} - \frac{1}{N}\boldsymbol{1}^T\boldsymbol{H}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H})^T \tag{39}$$

$$= \nabla_{\boldsymbol{\theta}_p}\Big[2\sum_{i=1}^{N}\big[\text{sg}[(\boldsymbol{H}_{i:}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H} - \frac{1}{N}\boldsymbol{1}^T\boldsymbol{H}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H})(\boldsymbol{X} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\boldsymbol{X})]f_p(\boldsymbol{x}^{(i)})\big]\Big](\boldsymbol{\theta}_c - \boldsymbol{\theta}_p). \tag{40}$$

The similarity can be approximated by Taylor expansion as

$$s(\bar{f}_p, f_c) = s(f_p, f_c)(1 + 2\frac{\langle \boldsymbol{H}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H}, \boldsymbol{H}(\boldsymbol{G}\boldsymbol{X}^T + \boldsymbol{X}\boldsymbol{G}^T)\boldsymbol{H}\rangle}{\langle \boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H}, \boldsymbol{H}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H}\rangle} - 2\frac{\langle \boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H}, \boldsymbol{H}(\boldsymbol{G}\boldsymbol{X}^T + \boldsymbol{X}\boldsymbol{G}^T)\boldsymbol{H}\rangle}{\langle \boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H}, \boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H}\rangle}) + \delta \tag{41}$$

$$= s(f_p, f_c)\big[1 + \nabla_{\boldsymbol{\theta}_p}[\sum_{i=1}^{N}\text{sg}(\zeta_i)f_p(\boldsymbol{x}^{(i)})](\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)\big] + \delta, \tag{42}$$

where

$$\begin{aligned}\zeta_i = &4\frac{(\boldsymbol{H}_{i:}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H} - \frac{1}{N}\boldsymbol{1}^T\boldsymbol{H}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H})(\boldsymbol{X} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\boldsymbol{X})}{\langle \boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H}, \boldsymbol{H}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H}\rangle} \\ &- 4\frac{(\boldsymbol{H}_{i:}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H} - \frac{1}{N}\boldsymbol{1}^T\boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H})(\boldsymbol{X} - \frac{1}{N}\boldsymbol{1}\boldsymbol{1}^T\boldsymbol{X})}{\langle \boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H}, \boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H}\rangle}\end{aligned} \tag{43}$$

$$= 4\frac{\boldsymbol{H}_{i:}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H}\boldsymbol{X}}{\langle \boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H}, \boldsymbol{H}\boldsymbol{Y}\boldsymbol{Y}^T\boldsymbol{H}\rangle} - 4\frac{\boldsymbol{H}_{i:}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H}\boldsymbol{X}}{\langle \boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H}, \boldsymbol{H}\boldsymbol{X}\boldsymbol{X}^T\boldsymbol{H}\rangle}. \tag{44}$$

3.6. Linear Approximation of Distance Correlation

The calculation of Distance Correlation can be written as

$$s(\bar{f}_p, f_c) = \frac{\langle \boldsymbol{H}\boldsymbol{D}_{\bar{\boldsymbol{X}}}\boldsymbol{H}, \boldsymbol{H}\boldsymbol{D}_{\boldsymbol{Y}}\boldsymbol{H}\rangle^2}{\langle \boldsymbol{H}\boldsymbol{D}_{\bar{\boldsymbol{X}}}\boldsymbol{H}, \boldsymbol{H}\boldsymbol{D}_{\bar{\boldsymbol{X}}}\boldsymbol{H}\rangle\langle \boldsymbol{H}\boldsymbol{D}_{\boldsymbol{Y}}\boldsymbol{H}, \boldsymbol{H}\boldsymbol{D}_{\boldsymbol{Y}}\boldsymbol{H}\rangle}, \tag{45}$$

where the distance matrices $\boldsymbol{D}_{\bar{\boldsymbol{X}}}$ and $\boldsymbol{D}_{\boldsymbol{Y}}$ are defined as following:

$$(\boldsymbol{D}_{\boldsymbol{X}})_{ij} = ||\boldsymbol{X}_{i:} - \boldsymbol{X}_{j:}||_2, \tag{46}$$

$$(\boldsymbol{D_Y})_{ij} = ||\boldsymbol{Y}_{i:} - \boldsymbol{Y}_{j:}||_2, \tag{47}$$

$$\boldsymbol{D}_{ij:} = \frac{\boldsymbol{X}_{i:} - \boldsymbol{X}_{j:}}{||\boldsymbol{X}_{i:} - \boldsymbol{X}_{j:}||_2}, \tag{48}$$

$$(\boldsymbol{D}_{\bar{\boldsymbol{X}}})_{ij} = ||\bar{\boldsymbol{X}}_{i:} - \bar{\boldsymbol{X}}_{j:}||_2 = (\boldsymbol{D_X})_{ij} + \boldsymbol{D}_{ij:}(\boldsymbol{G}_{i:} - \boldsymbol{G}_{j:}) \tag{49}$$

Note that the $\boldsymbol{D} \in \mathbb{R}^{N\times N\times K}$ defined above is a third-order tensor, whose indexing is similar to the previously used one for matrix. By substituting the definitions above into the Distance Correlation, its approximation can be obtained:

$$\begin{aligned} s(\bar{f}_p, f_c) = s(f_p, f_c)\Big[1 + \frac{4}{\langle \boldsymbol{HD_XH}, \boldsymbol{HD_YH}\rangle}\sum_{i=1}^{N}\Big[\sum_{j=1}^{N}\big[[(\boldsymbol{HD_YH})_{ij} - \frac{1}{N}\sum_{q=1}^{N}(\boldsymbol{HD_YH})_{jq}](\boldsymbol{D}_{ij:} - \frac{1}{N}\sum_{q=1}\boldsymbol{D}_{iq:})\big]\boldsymbol{G}_{i:}^T\Big] \\ - \frac{4}{\langle \boldsymbol{HD_XH}, \boldsymbol{HD_XH}\rangle}\sum_{i=1}^{N}\Big[\sum_{j=1}^{N}\big[[(\boldsymbol{HD_XH})_{ij} - \frac{1}{N}\sum_{q=1}^{N}(\boldsymbol{HD_XH})_{jq}](\boldsymbol{D}_{ij:} - \frac{1}{N}\sum_{q=1}\boldsymbol{D}_{iq:})\big]\boldsymbol{G}_{i:}^T\Big]\Big] + \delta \end{aligned} \tag{50}$$

$$= s(f_p, f_c)\big[1 + \nabla_{\boldsymbol{\theta}_p}[\sum_{i=1}^{N}\mathrm{sg}(\xi_i)f_p(\boldsymbol{x}^{(i)})](\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)\big] + \delta, \tag{51}$$

where

$$\begin{aligned} \xi_i = &\frac{4\sum_{j=1}^{N}\big[[(\boldsymbol{HD_YH})_{ij} - \frac{1}{N}\sum_{q=1}^{N}(\boldsymbol{HD_YH})_{jq}](\boldsymbol{D}_{ij:} - \frac{1}{N}\sum_{q=1}\boldsymbol{D}_{iq:})\big]}{\langle \boldsymbol{HD_XH}, \boldsymbol{HD_YH}\rangle} \\ &- \frac{4\sum_{j=1}^{N}\big[[(\boldsymbol{HD_XH})_{ij} - \frac{1}{N}\sum_{q=1}^{N}(\boldsymbol{HD_XH})_{jq}](\boldsymbol{D}_{ij:} - \frac{1}{N}\sum_{q=1}\boldsymbol{D}_{iq:})\big]}{\langle \boldsymbol{HD_XH}, \boldsymbol{HD_XH}\rangle} \end{aligned} \tag{52}$$

$$= \frac{4\sum_{j=1}^{N}[(\boldsymbol{HD_YH})_{ij}\boldsymbol{D}_{ij:}]}{\langle \boldsymbol{HD_XH}, \boldsymbol{HD_YH}\rangle} - \frac{4\sum_{j=1}^{N}[(\boldsymbol{HD_XH})_{ij}\boldsymbol{D}_{ij:}]}{\langle \boldsymbol{HD_XH}, \boldsymbol{HD_XH}\rangle} \tag{53}$$

$$= \frac{4\boldsymbol{H}_{i:}\boldsymbol{D_YHD}_{i::}}{\langle \boldsymbol{HD_XH}, \boldsymbol{HD_YH}\rangle} - \frac{4\boldsymbol{H}_{i:}\boldsymbol{D_XHD}_{i::}}{\langle \boldsymbol{HD_XH}, \boldsymbol{HD_XH}\rangle} \tag{54}$$

3.7. Connection Between Linearization and Minimal Distance

In this subsection, we solve the two optimizations in Proposition 2 in the main text. For the first maximization problem, $\boldsymbol{W}(\boldsymbol{x})$ can be solved for each $\boldsymbol{W}$.

$$\max_{\boldsymbol{W}(\boldsymbol{x})} s(f_p'(\boldsymbol{x}), f_c(\boldsymbol{x})) = \max_{\boldsymbol{W}(\boldsymbol{x})} -\frac{1}{K}||f_p'(\boldsymbol{x}) - f_c(\boldsymbol{x})||_2^2 \max_{\boldsymbol{W}(\boldsymbol{x})} -\frac{1}{K}||f_p(\boldsymbol{x}) + \boldsymbol{W}(\boldsymbol{x})(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) - f_c(\boldsymbol{x})||_2^2. \tag{55}$$

Taking the derivative, the optimal condition is

$$\boldsymbol{0} = (f_p(\boldsymbol{x}) + \boldsymbol{W}(\boldsymbol{x})(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p) - f_c(\boldsymbol{x}))(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)^T. \tag{56}$$

Thus,

$$\boldsymbol{W}(\boldsymbol{x}) = (f_c(\boldsymbol{x}) - f_p(\boldsymbol{x}))(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)^T[(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)^T]^{-1}. \tag{57}$$

Because f_c and f_p are real parent-child pair, f_c can be approximated by the Linearized model, in other words, f_c can be estimated by the Taylor expansion at f_p. Thus,

$$\boldsymbol{W}(\boldsymbol{x}) = \nabla_{\boldsymbol{\theta}_p} f_c(\boldsymbol{x})(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)^T[(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)^T]^{-1} + \delta \approx \nabla_{\boldsymbol{\theta}_p} f_c(\boldsymbol{x}). \tag{58}$$

Similarly, the optimal condition for the second optimization is

$$\sum_{i=1}^{N}[\nabla_{\boldsymbol{\theta}_p} f_p(\boldsymbol{x}^{(i)})]^T\nabla_{\boldsymbol{\theta}_p} f_p(\boldsymbol{x}^{(i)})\boldsymbol{Z} = \sum_{i=1}^{N}[\nabla_{\boldsymbol{\theta}_p} f_c(\boldsymbol{x}^{(i)})]^T[f_c(\boldsymbol{x}^{(i)}) - f_p(\boldsymbol{x}^{(i)})] \tag{59}$$

By Taylor expansion, the right-hand side can be rewritten as $\sum_{i=1}^{N}[\nabla_{\boldsymbol{\theta}_p} f_p(\boldsymbol{x}^{(i)})]^T\nabla_{\boldsymbol{\theta}_p} f_p(\boldsymbol{x}^{(i)})(\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)$. Thus, $\boldsymbol{Z} = (\boldsymbol{\theta}_c - \boldsymbol{\theta}_p)$.

4. Further Discussions

4.1. The Applicability of Linearized Network

Our approach does not align perfectly with the NTK setting, as the NTK method employs Gaussian initialization for the neural network parameters. In contrast, our model's initialization stems from a pre-training phase. Given that [2] demonstrated that the influence of gradient descent on individual parameter values is minimal, and it primarily affects the collective behavior of parameters, i.e., the features and outputs of the neural network, we anticipate that parameters initialized with a Gaussian distribution will still approximately follow a Gaussian distribution after gradient descent training. Consequently, we can continue to leverage the relevant results from the NTK framework.

4.2. Potential Application

Lineage detection identifies connections between child models and ancestors, enabling the detection/classification of inherited failure modes. For example, given some parent models and a child model, suppose each parent has a certain failure mode, lineage detection identifies the child's most probable parent or determines the probability of belonging to each parent, thereby assessing which failure mode the child is likely to inherit or the probability of inheriting various failure modes. The child model can then be cleansed using unlearning or debiasing methods. In fact, this process can also be reversed: upon observing certain failure modes in a child model, lineage detection probabilities can be used to evaluate the likelihood of each parent model having these failure modes in a Bayesian way, leading to targeted cleansing of the parent models.

4.3. Limitations and Future Explorations

As the very first exploration in the thread of model lineage detection, we have endeavored to encompass a diverse range of computer vision tasks, including classification, segmentation, and detection. We have delved into various learning paradigms, from direct fine-tuning and multi-stage fine-tuning to continual learning and knowledge distillation. However, numerous settings and deep learning tasks remain outside the scope of this paper. For instance, when dealing with dynamic neural networks, the challenge arises of concurrently addressing lineage prediction and the variability in model parameter space dimensions. This limitation primarily arises from our method's requirement that the parent model and child models be naturally aligned. We leave lineage detection with architecture modifications as future research work. Furthermore, although our lineage detector can also be applied in settings that only utilize neural network outputs, the task we addressed in this work is more akin to a white-box setup, if applying the taxonomy in the field of adversarial learning, requiring knowledge of both the network's parameters and its feature outputs. This setup is justified, as the finetuning process inherently involves complex interactions between parameters and output features. Indeed, the prevalence of models developed based on open-source projects also presents a viable application scenario for our white-box approach to lineage detection. However, compared to a black-box setup, our task can be relatively easier. As the very first work for lineage detection, investigating this ideal white-box scenario is indispensable. We hope our work will contribute to a deeper understanding of lineage detection tasks and serve as a foundation for future research. We leave the exploration of black-box lineage detection and other more challenging lineage detection tasks for future works.

References

[1] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819, 2021. 6

[2] Arthur Jacot, Clément Hongler, and Franck Gabriel. Neural tangent kernel: Convergence and generalization in neural networks. In *NeurIPS*, 2018. 13

[3] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 2017. 6

[4] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models`, 2019. 4