Journal of Machine Learning Research 23 (2022) 1-47 Submitted 10/21; Revised 3/22; Published 6/22

Ranking and Tuning Pre-trained Models: A New Paradigm for Exploiting Model Hubs

Kaichao You[1] * youkaichao@gmail.com
Yong Liu[1] * liuyong21@mails.tsinghua.edu.cn
Ziyang Zhang[2] zhangziyang11@huawei.com
Jianmin Wang[1] jimwang@tsinghua.edu.cn
Michael I. Jordan[3] jordan@cs.berkeley.edu
Mingsheng Long[1] † mingsheng@tsinghua.edu.cn

[1] *School of Software, BNRist, Tsinghua University, Beijing 100084, China.*
[2] *Advanced Computing and Storage Lab, Huawei Technologies Co. Ltd*
[3] *Division of Computer Science and Department of Statistics, UC Berkeley, CA 94720-1776, USA*

Editor: Stefan Harmeling

Abstract

Model hubs with many pre-trained models (PTMs) have become a cornerstone of deep learning. Although built at a high cost, they remain *under-exploited*—practitioners usually pick one PTM from the provided model hub by popularity and then fine-tune the PTM to solve the target task. This naïve but common practice poses two obstacles to full exploitation of pre-trained model hubs: first, the PTM selection by popularity has no optimality guarantee, and second, only one PTM is used while the remaining PTMs are ignored. An alternative might be to consider all possible combinations of PTMs and extensively fine-tune each combination, but this would not only be prohibitive computationally but may also lead to statistical over-fitting. In this paper, we propose a new paradigm for exploiting model hubs that is intermediate between these extremes. The paradigm is characterized by two aspects: (1) We use an evidence maximization procedure to estimate the maximum value of label evidence given features extracted by pre-trained models. This procedure can rank all the PTMs in a model hub for various types of PTMs and tasks *before fine-tuning*. (2) The best ranked PTM can either be fine-tuned and deployed if we have no preference for the model's architecture or the target PTM can be tuned by the top K ranked PTMs via a Bayesian procedure that we propose. This procedure, which we refer to as *B-Tuning*, not only improves upon specialized methods designed for tuning homogeneous PTMs, but also applies to the challenging problem of tuning heterogeneous PTMs where it yields a new level of benchmark performance.

Keywords: Pre-trained Model Hub, Model Ranking, Model Tuning, Transfer Learning

1. Introduction

Deep neural networks (He et al., 2015, 2016; Devlin et al., 2019) trained on large-scale datasets (Deng et al., 2009; Russakovsky et al., 2015; Merity et al., 2017) and specialized computational devices (Jouppi et al., 2017) have achieved striking, human-level perfor-

∗. The first two authors contributed equally to the paper.
†. Mingsheng Long is the corresponding author.

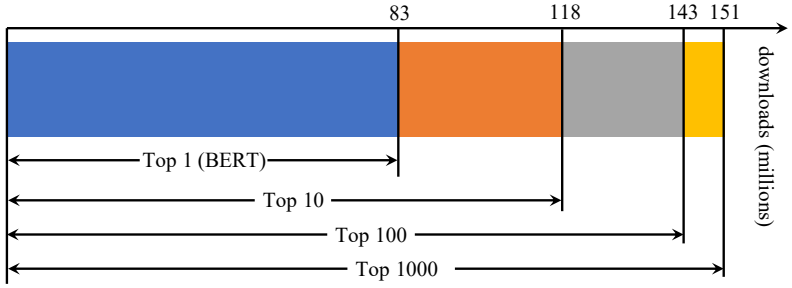


Figure 1: Monthly download statistics (in millions) for top popular models in the HuggingFace Transformer library. The most popular PTM (BERT) takes up more than a half downloads, while other PTMs are much less exploited.

mance on many pattern recognition tasks in both computer vision and natural language processing. Moreover, research has shown that deep neural networks trained on large-scale pre-training tasks (Yang et al., 2019; Clark et al., 2020; Brown et al., 2020) can produce generic representations (Donahue et al., 2014) that benefit downstream tasks such as object detection (Girshick et al., 2014) and language understanding (Wang et al., 2019). These trained neural networks are known as pre-trained models (PTMs). Readers can refer to dedicated surveys (Han et al., 2021; Qiu et al., 2020; Bommasani et al., 2021) for a holistic overview of pre-trained models. The power (Brown et al., 2020) of PTMs, together with the transfer learning paradigm of "pre-training $\rightarrow$ fine-tuning" to exploit PTMs, has had significant impact in both vision (Kornblith et al., 2019) and language (Devlin et al., 2019), and the influence of PTMs is growing in nearby communities such as geometric learning (Hu et al., 2020).

The cost of training PTMs varies from hundreds of GPU hours (He et al., 2016) to hundreds of GPU days (Devlin et al., 2019), which can be prohibitively expensive for individual researchers and academic labs. Most pre-trained models are provided by central repositories, including PyTorch Hub,[1] TensorFlow Hub,[2] and HuggingFace Transformer Models.[3] Such collections of pre-trained models are called a "pre-trained model hubs" (PTM hubs), and they have become very popular; for example, the HuggingFace Transformer library (Wolf et al., 2020), the most popular BERT model (Devlin et al., 2019) is currently being downloaded over 80 million times every month.

Although centralized repositories spend enormous resources to provide large-scale PTM hubs to the public, it turns out that practitioners often pick the most popular PTM, meaning that the whole PTM hub is insufficiently exploited. Figure 1 analyzes the monthly downloads of PTMs in the HuggingFace Transformer hub. Beyond several popular models the remaining PTMs in the hub are seldom downloaded. The statistics in PyTorch Hub and TensorFlow Hub are essentially the same—several popular PTMs dominate the rest.

Naïvely picking the most popular PTM is far from optimal in two respects: (1) The PTM selection is *task-specific* and one PTM cannot be optimal for all the tasks; different tasks

1. `https://pytorch.org/hub/`
2. `https://www.tensorflow.org/hub`
3. `https://huggingface.co/models`

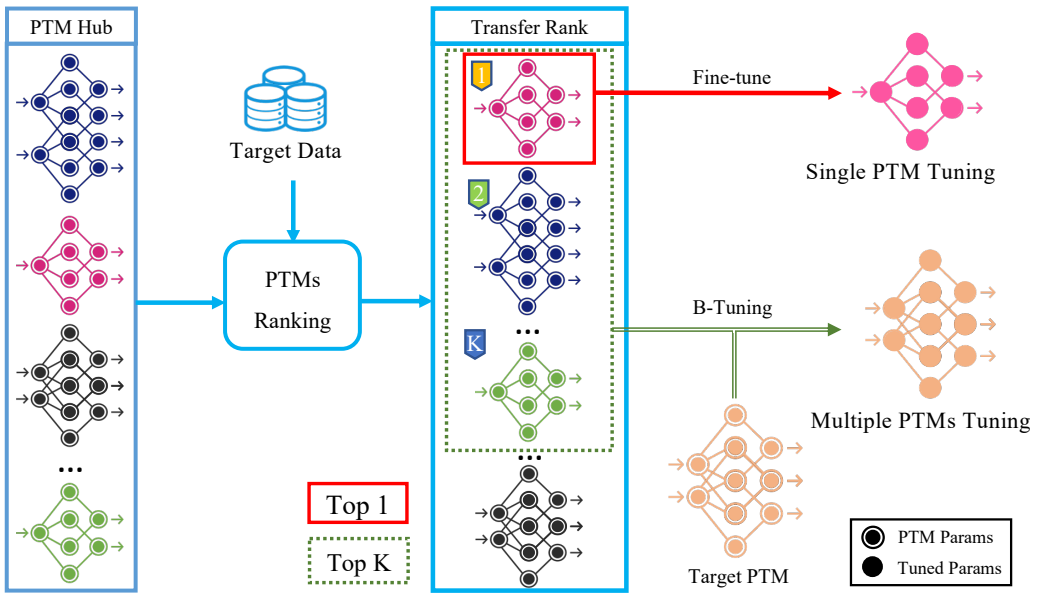


Figure 2: The proposed paradigm of ranking and tuning pre-trained models. PTMs are ranked by their transferability w.r.t. the target data, then either the best PTM is fine-tuned, or the target PTM is tuned by top-K PTMs via the proposed B-Tuning.

generally favor different PTMs, depending on the compatibility between the pre-trained model and the target task (You et al., 2021). (2) Only one PTM is exploited, and opportunities to obtain benefits from ensembling or aggregation are put aside. Correspondingly, there are two reasons why practitioners resort to the suboptimal naïve practice: (1) maximally exploiting a PTM hub requires trying all combinations of PTMs and extensively fine-tuning each PTM combination, which requires unaffordable computation; (2) even if the significant computational cost can be paid, it is unclear how to exploit multiple PTMs in transfer learning. As discussed in Section 2.4, Shu et al. (2021) studied the problem in a limited case, but a general solution is lacking.

To fully exploit PTM hubs, we propose a new paradigm: ranking and tuning pre-trained models. Figure 2 provides an overview of the paradigm. It consists of two parts: (1) PTMs are *ranked* by a transferability metric; (2) top-ranked PTMs are *tuned* to meet downstream applications' requirements. Our preliminary work (You et al., 2021) proposed a method that we referred to as "logarithm of maximum evidence" (LogME) to estimate the compatibility between PTMs and downstream datasets. We demonstrated its effectiveness on a variety of PTMs. With an effective transferability rank, the best ranked PTM can be fine-tuned if there are no constraints on network architecture like inference time or hardware-friendly operators. If these constraints are present, the qualified PTM with desired architecture might not be the best-ranked one, but it can be tuned by top-K ranked PTMs via a novel B-Tuning algorithm as discussed in Section 5.3.

Compared with picking the most popular PTM, our proposed new paradigm features two significant advantages: (1) it provides a *task-adaptive ranking* of all PTMs in a PTM

hub to enable optimal selection of PTMs; (2) it opens the new possibility to *exploit multiple PTMs for tuning*, breaking the stereotype that fine-tuning must be tied up with a single PTM. The new paradigm can be useful in a broad variety of scenarios, as pre-trained models are increasingly important in deep learning.

Besides a new paradigm of exploiting PTM hubs, this paper brings novel theoretical analyses and a new algorithm for multiple PTM tuning. (1) On the theoretical side, we derive a sufficient condition for the evidence maximization algorithm (MacKay, 1992) to converge and analyze the influence of dimensionality on LogME. The evidence maximization algorithm (MacKay, 1992) has been used primarily as a heuristic; a rigorous convergence condition has been lacking. (2) On the algorithm design side, we devise a method that we refer to as "B-Tuning" for tuning multiple PTMs using Bayesian learning. We show that this method surpasses the dedicated method (Shu et al., 2021) for homogeneous PTMs (PTMs with the same architecture) and also works for the challenging scenario with heterogeneous PTMs (PTMs with different architectures).

The contributions of this paper are summarized as follows:

1. We propose a new paradigm for exploiting PTM hubs, namely ranking and tuning pre-trained models. It has significant advantages compared with the common practice of naïvely fine-tuning a popular pre-trained model.
2. Concerning ranking PTMs, we propose LogME for transferability assessment and develop a fast algorithm to accelerate the computation. LogME is easy to interpret and is extremely efficient: it brings roughly 3700× speedup in wall-clock time and requires just 1% memory footprint compared with brute-force fine-tuning. Theoretical analyses confirm the rationality of LogME, and lay a theoretical foundation for a heuristic algorithm in evidence maximization.
3. For tuning PTMs, two possible scenarios are studied. In the academic scenario without specific requirements for the PTM architecture, the best ranked pre-trained model according to the transferability rank can be selected for subsequent fine-tuning; in the industrial scenario where a specific PTM architecture is required to meet the budget of computation and energy, we propose B-Tuning to tune the given pre-trained model with top-K ranked PTMs, even though these PTMs are heterogeneous.

Compared with our conference paper (You et al., 2021) that only proposed LogME for transferability estimation, this paper extends LogME to a paradigm of ranking and tuning pre-trained models. Additional theoretical analyses are available in the ranking part, and a new algorithm is presented in the tuning part. Moreover, LogME is tested against additional tasks like named entity recognition (Sang and De Meulder, 2003) in Section 6.2.5 and prompt learning (Liu et al., 2021a) in Section 6.6.

We will need to make significant use of notation to describe our ideas; thus, for the convenience of readers, all of the notation is collected in Table 9.

The basic problem setup contains a PTM hub with M pre-trained models, $\{\phi_k\}_{k=1}^M$, and the transfer learning task is given by a labeled dataset, $\mathcal{D} = \{(x_i, Y_i)\}_{i=1}^n$, with n labeled data points. This paper focuses on classification and regression tasks, so the label $Y_i \in \mathbb{R}^C$ is C dimensional.

The rest of the paper is organized as follows: Section 2 summarizes related work, Section 3 focuses on ranking and describes the LogME transferability metric, Section 4 presents theoretical analyses for LogME, Section 5 focuses on tuning and introduces the

B-Tuning method for multiple PTM tuning, Section 6 presents all the experiments, and finally Section 7 concludes the paper.

2. Related Work

2.1 Transfer learning

Transfer learning (Thrun and Pratt, 1998) consists of transductive transfer, inductive transfer, task transfer, and so on. A well-known transductive paradigm is domain adaptation (Quionero-Candela et al., 2009), which aims for reducing domain shifts by transferring samples, hidden features (Long et al., 2015; Ganin and Lempitsky, 2015), and categorical information (Cao et al., 2022). Inductive transfer, in particular fine-tuning in deep learning (Erhan et al., 2010; Yosinski et al., 2014), exploits prior knowledge (pre-trained models) to improve the performance of target tasks. Task transfer learning (Zamir et al., 2018) focuses on how to transfer tasks rather than pre-trained models. It aims to discover shared relevance among tasks (Ben-David and Schuller, 2003) and to exploit the relationship for improvement on the target task. In the context of deep learning, transfer learning usually means inductive transfer with a pre-trained model, which is the focus of this paper.

Many previous works (Yosinski et al., 2014; Kornblith et al., 2019; Neyshabur et al., 2020) have shown the benefit of initializing a deep neural network with a pre-trained model. Apart from the vanilla method (*i.e.*, the pre-trained model is just used for initialization), researchers have recently proposed sophisticated fine-tuning techniques like regularization (Li et al., 2018; Chen et al., 2019), additional supervision (You et al., 2020), and carefully designed architectures (Kou et al., 2020). They can further improve transfer learning performance, but empirically *these fine-tuning methods do not change the ranking of pre-trained models on downstream tasks*. That is, if pre-trained model A is better than pre-trained model B after vanilla fine-tuning, empirically A is better than B when those advanced techniques are integrated. For example, on three datasets and four sampling rates in Table 2 of You et al. (2020), better fine-tuning performance primarily indicates better Co-Tuning (their proposed method) performance, implying that the transferability of a pre-trained model might be *task-specific* rather than *method-specific*. Therefore, our experiments stick to the vanilla fine-tuning during PTM ranking.

2.2 PTMs and PTM hubs

Pre-trained models (PTMs) are generalizable deep networks trained on large-scale data. They can be transferred to a series of downstream tasks. They have become a cornerstone in deep learning and sometimes are known as foundation models (Bommasani et al., 2021). Typical categories of PTMs are summarized in the following.

Supervised PTMs. In the ImageNet classification challenge, He et al. (2015) developed the first deep neural network that surpassed human performance. By supervised pre-training on the ImageNet dataset, deep models marched towards higher accuracy, fewer parameters, and lower computation. InceptionNet (Szegedy et al., 2015) made use of parallel convolutional filters to extract different levels of features. ResNet (He et al., 2016) introduced skip-connections to ease the vanishing gradient problem so that much deeper networks could be trained. Inspired by ResNet, DenseNet (Huang et al., 2017) was equipped with dense

skip-connections to reuse features in a parameter-efficient manner. MobileNet (Sandler et al., 2018) was a low-parameter, mobile-friendly network structure which was further optimized with the help of network architecture search to become MNASNet (Tan et al., 2019).

Unsupervised PTMs. Although supervised pre-training is the most common practice, the labeling cost of large-scale data can be prohibitively expensive. As a large amount of unlabeled data on the Internet are available but under-exploited, recently many researchers have sought to apply self-supervised learning (Jing and Tian, 2020) on unlabeled data (Mahajan et al., 2018) with contrastive loss (Gutmann and Hyvärinen, 2010). Accordingly, a family of unsupervised deep models has emerged in recent years. He et al. (2020) proposed Momentum Contrast with a creative queue structure to fully exploit the manifold structure of unlabeled data. Chen et al. (2020a) significantly improved performance by exploring data augmentation, multi-layer projection head, and empirical designs. Designing better strategies for contrastive pre-training is still under active research (Tian et al., 2020).

Language PTMs. In recent years, natural language processing has been revolutionized by language PTMs. Unsupervised pre-trained models have been well established by training masked language models (Devlin et al., 2019) or autoregressive language models (Yang et al., 2019) on large unlabeled corpora (Merity et al., 2017). Liu et al. (2019) explored many practical aspects of the training of language models. Sanh et al. (2019) proposed distillation to make PTMs smaller and faster. These pre-trained language models are very common in winning submissions on important benchmarks like GLUE (Wang et al., 2018) and SQuAD (Rajpurkar et al., 2016), and have established their profound influence in the industry.

PTMs are grouped together to be hosted in PTM hubs like TorchVision and HuggingFace Models. Industry labs have invested significant resources in training these PTMs, but unfortunately, PTM hubs are under-exploited, as quantitatively measured in Figure 1 and described in the introduction section. The goal of this paper is to develop a new paradigm of exploiting PTM hubs, so that pre-trained models can be more widely exploited.

2.3 Assessing the transferability of pre-trained models

Assessing the transferability of PTMs has great significance in guiding the practice of deep learning. It can be used to rank available PTMs and act as a criterion for pre-trained model selection. Yosinski et al. (2014) studied the performance of transferring different layers of a pre-trained model, and Kornblith et al. (2019) studied a wide variety of ImageNet PTMs with modern network architectures. These papers aim for a deep understanding (Neyshabur et al., 2020) of transfer learning by expensive and exhaustive fine-tuning with major computation cost (see Section 6.5), which is hard for practitioners to afford. In most scenarios, practitioners care most about PTMs' relative ranking on target tasks to guide PTM selection, requiring a practical assessment method that is *efficient*, *accurate*, and *general*: a transferability assessment method should be efficient enough compared with brute-force fine-tuning (Zamir et al., 2018), should be accurate enough to identify potentially best models, and should be general enough to tackle a wide variety of common scenarios.

LEEP (Nguyen et al., 2020) and NCE (Tran et al., 2019) were the first two methods to assess the transferability of pre-trained models. Nguyen et al. (2020) constructed an empirical predictor from the joint distribution $p(y_t, y_s)$ over pre-trained labels y_s and target

labels y_t, and calculated the log expectation of the empirical predictor (LEEP) as the transferability measure. The empirical predictor predicts the probability of the target class y_t as $\sum_{y_s \in \mathcal{Y}_s} p(y_t|y_s)p(y_s)$, where $p(y_s)$ comes from the PTM's prediction over pre-trained categories. Negative Conditional Entropy (NCE) proposed by Tran et al. (2019) depended on an information-theoretic quantity (Cover, 1999) to reveal the transferability and hardness between different tasks. It estimated the joint distribution $p(y_t, y_s)$ with one-hot labels and predictions, and defined NCE as $-H(y_t|y_s)$, *i.e.*, the negative conditional entropy of target labels y_t given PTM's predictions y_s.

Table 1: Applicability of existing methods and LogME proposed in this paper.

Modality	Pre-train	Target	Method		
			LEEP	NCE	LogME
vision	classification	classification	✓	✓	✓
	classification	regression	✗	✗	✓
	contrastive	classification	✗	✗	✓
	contrastive	regression	✗	✗	✓
language	language modeling	classification	✗	✗	✓

All of these methods, however, had their limitations. As shown in Table 1, they can only handle classification tasks with supervised pre-trained models. Increasingly popular contrastive pre-trained models and language models are out of their scope. The LogME algorithm proposed in this paper extends the applicability of transferability assessment to these cases. LogME is fast to compute, less prone to over-fitting, and broadly applicable to various pre-trained models/downstream tasks/data modalities. Its performance is validated by extensive experiments. Prior to this paper, for most (four out of five) transfer learning settings, task adaptive transferability assessment did not have a satisfactory solution. In addition, LogME's statistical rigor makes it extensible to multiple PTM tuning (see Section 5.3), which further fleshes out the new paradigm of ranking and tuning pre-trained models.

2.4 Multiple PTM tuning

A straightforward approach in transfer learning is to fine-tune models initialized from pre-trained parameters, which we call "*single PTM tuning*" because it can only exploit a specific pre-trained model during fine-tuning.

It is widely acknowledged that the success of transfer learning comes from the knowledge in the pre-trained model. Considering that there are so many PTMs in a PTM hub, it is appealing to transfer multiple PTMs simultaneously, a problem we call "*multiple PTM tuning*." We might expect multiple PTM tuning to outperform single PTM tuning.

Unfortunately, multiple PTM tuning is under-explored due to technical challenges. If multiple PTMs are homogeneous, *i.e.*, they share the same network architecture, the problem becomes easier. Researchers in this area focused on how to align and merge multiple homogeneous PTMs. Singh and Jaggi (2020) defined transportation cost between neural representations and minimized the induced Wasserstein distance to align neurons from

each PTM. Shu et al. (2021) developed a channel-wise alignment method dedicated to convolutional neural networks with a learnable gating function to merge multiple PTMs. Prior to this paper, Shu et al. (2021) held the state-of-the-art result in homogeneous PTMs tuning.

Heterogeneous PTMs tuning is much more difficult than homogeneous PTMs tuning and it is still unresolved how to address this general form of tuning. In practice, pre-trained models in PTM hubs generally have different architectures and it has become increasingly urgent to address the heterogeneity problem.

This paper proposes a methodology for exploiting general PTM hubs. In the proposed paradigm, PTMs are first ranked by LogME, then the top-K ranked PTMs from the PTM hub are selected for multiple PTM tuning. A Bayesian tuning method (B-Tuning, see Section 5.3) is further proposed to solve the multiple PTM tuning problem. Overall the method proposes a solution to the heterogeneous PTMs tuning problem, and it surpasses the state-of-the-art method (Shu et al., 2021) dedicated to homogeneous PTM tuning.

3. Ranking Pre-Trained Models

The ranking of pre-trained models requires a transferability metric. But before introducing the transferability metric in Section 3.2, we discuss how to quantify its fidelity to the reference transferability performance, which is elaborated in the following Section 3.1.

3.1 How to measure the performance of a transferability metric?

A transfer learning task (in the form of a dataset $\mathcal{D} = \{(x_i, Y_i)\}_{i=1}^n$) should have an evaluation metric (accuracy, MAP, MSE, *etc.*) to measure the reference transfer performance T_k of fine-tuning ϕ_k with sufficient hyper-parameter tuning. A practical assessment method should produce a score S_k for each pre-trained model ϕ_k (ideally without fine-tuning ϕ_k on $\mathcal{D}$), and the scores $\{S_k\}_{k=1}^M$ should well correlate with $\{T_k\}_{k=1}^M$ so that top-performing pre-trained models can be selected by simply evaluating the scores $\{S_k\}_{k=1}^M$.

A perfect pre-trained model assessing method would produce $\{S_k\}_{k=1}^M$ with precisely the same order as $\{T_k\}_{k=1}^M$. To measure the deviation from the perfect method, we can use simple metrics like top-1 accuracy or top-K accuracy (whether the fraction among top-K in $\{S_k\}_{k=1}^M$ are also top-K in $\{T_k\}_{k=1}^M$). Nevertheless, top-1 accuracy is too conservative and top-K accuracy is not comparable across different values of M. Rank correlation (Fagin et al., 2003) is a good alternative to directly measure the correlation between $\{S_k\}_{k=1}^M$ and $\{T_k\}_{k=1}^M$. Prior work (Nguyen et al., 2020) adopted Pearson's linear correlation coefficient, but neither Pearson's linear correlation nor its variant (Spearman's rank correlation) has a simple interpretation (see the interpretation of τ below). Therefore, they are not used in this paper.

The rank correlation method we choose is Kendall's τ coefficient (Kendall, 1938), which counts concordant pairs to capture the possibility of T_i *being better than* T_j *if* S_i *is better than* S_j in choosing a good pre-trained model.

Without loss of generality, we assume larger values of transfer performance T and score S are preferred (*e.g.*, accuracy). If this is not the case (*e.g.*, transfer performance is measured by mean square error and small values are favored), the negation $-T$ can be considered. For a pair of measures (T_i, S_i) and (T_j, S_j), the pair is concordant if $T_i < T_j \wedge S_i < S_j$ or

$T_i > T_j \wedge S_i > S_j$ (concisely speaking, $\text{sgn}(T_i - T_j)\text{sgn}(S_i - S_j) = 1$). Kendall's τ coefficient is defined by the following equation, which enumerates all $\binom{M}{2}$ pairs and counts the number of concordant pairs minus the number of discordant pairs.

$$\tau = \frac{\sum_{1 \le i < j \le M} \text{sgn}(T_i - T_j)\text{sgn}(S_i - S_j)}{\binom{M}{2}}.$$

How to interpret τ (Fagin et al., 2003): The range of τ is $[-1, 1]$. $\tau = 1$ means T and S are perfectly correlated ($S_i > S_j \iff T_i > T_j$), and $\tau = -1$ means T and S are inversely correlated ($S_i > S_j \iff T_i < T_j$). *If T and S have a correlation value of τ, the probability of $T_i > T_j$ is $\frac{\tau+1}{2}$ when $S_i > S_j$.*

Pay attention to top-performing models. Since a major application of transferability metric is to select top-performing pre-trained models, discordant/concordant pairs should be weighted more if T_i, T_j, S_i, S_j are larger. This can be taken care of by τ_w (Vigna, 2015), a weighted variant of Kendall's τ. The details of calculating τ_w can be found in the SciPy implementation. With the weighting scheme, correlation value τ_w corresponds to a proportion interval of concordant pairs rather than a unique proportion value $\frac{\tau_w+1}{2}$. Nevertheless, the interval lies near the value $\frac{\tau_w+1}{2}$. Therefore, we can roughly use the probability $\frac{\tau_w+1}{2}$ of concordant pairs to interpret the correlation value τ_w.

In short, we measure the correlation between $\{S_k\}_{k=1}^M$ and $\{T_k\}_{k=1}^M$ by τ_w (Vigna, 2015). *Larger τ_w indicates a better correlation and better assessment.*

3.2 The LogME approach

This section describes LogME in detail. Since a transferability metric measures the transferability of pre-trained models, it should produce a score S_k for each PTM ϕ_k independent of the remaining PTMs. We thus drop the subscript k in this section.

An important goal of designing transferability metrics is to quickly assess many PTMs. With that in mind, we set the minimization of assessment time as a priority. First, to avoid expensive optimization of the whole PTM, PTM ϕ is regarded as a fixed feature extractor. Note that Nguyen et al. (2020) were limited to supervised pre-trained models because they used a pre-trained classification head h. In contrast, *we only use the pre-trained representation model ϕ so that the proposed method can be applied to any pre-trained model* (whether supervised pre-trained or unsupervised pre-trained).

With ϕ fixed, features $\{f_i = \phi(x_i)\}_{i=1}^n$ and labels $\{Y_i\}_{i=1}^n$ of the target task are the ingredients we can use to assess pre-trained models. The rest of this section discusses how to estimate the compatibility of features and labels as a transferability metric.

3.2.1 Evidence calculation

We first consider a simple case with D-dimensional features $f_i \in \mathbb{R}^D$ and scalar labels $y_i \in \mathbb{R}$. Note that the actual label Y_i can be non-scalar, and the way in which we extend from scalar labels y_i to vector labels Y_i is explained in Section 3.2.2.

Let the feature matrix $F \in \mathbb{R}^{n \times D}$ denote all the features and $y \in \mathbb{R}^n$ denote all the labels. A direct measurement of the compatibility between features F and labels y is the probability density $p(y|F)$, which is intractable without a parameterized model. Since the

rule-of-thumb transfer learning practice is to add a linear layer on top of the pre-trained model, we use a linear model upon features parameterized by w.

A straightforward approach to deal with the linear model is to find the best w^* by logistic or linear regression under maximum likelihood estimation, and to assess pre-trained models by the likelihood $p(y|F, w^*)$. However, it is well known that *maximum likelihood estimation is prone to over-fitting* (Bishop, 2006). Regularization techniques like ℓ_2-regularization may alleviate over-fitting at the cost of additional hyper-parameters, which requires manual intervention or grid search to tune those hyper-parameters. Even after extensive hyper-parameter tuning, its performance is not satisfying as observed in Section 6.6, because finding an optimal hyper-parameter is very difficult. *Ideally, a transferability metric should have no hyper-parameters so that it can be applied to downstream tasks without manual intervention.* Obviously, this approach does not satisfy the hyper-parameter-free property.

The disadvantage of the above approach can be overcome by the evidence approach introduced below. Evidence (also known as marginalized likelihood) is defined as $p(y|F) = \int p(w)p(y|F, w)\mathrm{d}w$, which integrates over all possible values of w rather than taking one w^* value. This evidence-based approach is an elegant model selection approach and has a rigorous theoretical foundation (Knuth et al., 2015). $p(w)$ and $p(y|F, w)$ are modeled by a graphical model (Figure 3) specified by two positive hyper-parameters α and β: the prior distribution of the weight is an isotropic multivariate Gaussian $w \sim \mathcal{N}(0, \alpha^{-1}I)$, and the distribution of each observation is a one-dimensional normal distribution $p(y_i|f_i, w, \beta) \sim \mathcal{N}(y_i|w^T f_i, \beta^{-1})$. Fortunately, hyper-parameters α and β can be automatically set to their optimal values as described in Section 3.2.2.

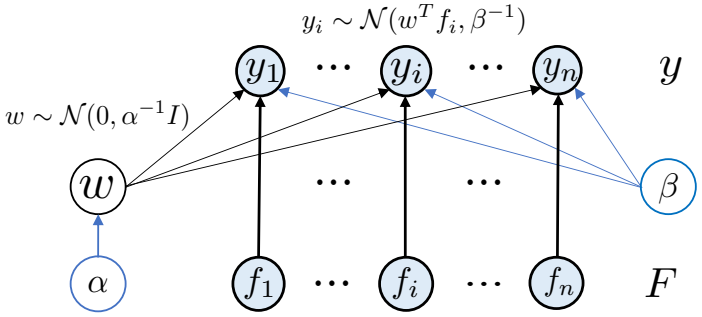


Figure 3: The directed graphical model for calculating evidence.

According to the causal structure in Figure 3 and the basic principles in graphical models (Koller and Friedman, 2009), the evidence can be calculated analytically as follows:

$$p(y|F, \alpha, \beta) = \int p(w|\alpha) \prod_{i=1}^{n} p(y_i|f_i, w, \beta)\mathrm{d}w = (\tfrac{\beta}{2\pi})^{\frac{n}{2}} (\tfrac{\alpha}{2\pi})^{\frac{D}{2}} \int e^{-\frac{\alpha}{2}w^T w - \frac{\beta}{2}||Fw-y||^2}\mathrm{d}w. \quad (1)$$

Equation 1 can be simplified by the identity $\int e^{-\frac{1}{2}(w^T A w + b^T w + c)}\,\mathrm{d}w = \sqrt{\frac{(2\pi)^D}{|A|}} e^{-\frac{1}{2}c + \frac{1}{8}b^T A^{-1} b}$. Taking the logarithm for simplicity, Equation 2 shows the logarithm of the evidence $\mathcal{L}$ as a function of α, β, where $A = \alpha I + \beta F^T F, m = \beta A^{-1} F^T y$.

$$\mathcal{L}(\alpha, \beta) = \log p(y|F, \alpha, \beta) = \tfrac{n}{2}\log\beta + \tfrac{D}{2}\log\alpha - \tfrac{n}{2}\log 2\pi - \tfrac{\beta}{2}||Fm - y||_2^2 - \tfrac{\alpha}{2}m^T m - \tfrac{1}{2}\log|A|. \quad (2)$$

3.2.2 Evidence maximization and LogME

An unresolved issue in Equation 2 is how to choose α, β. Gull (1989) suggested choosing α, β to maximize the evidence, *i.e.*, use $(\alpha^*, \beta^*) = \arg\max_{\alpha,\beta} \mathcal{L}(\alpha, \beta)$. Because m and A are coupled, it is a difficult problem to directly maximize $\mathcal{L}(\alpha, \beta)$. To address this, MacKay (1992) proposed a heuristic algorithm to solve the maximization problem: (1) set initial value of α, β; (2) evaluate A, m, γ with given α, β: $A = \alpha I + \beta F^T F, m = \beta A^{-1} F^T y, \gamma = \sum_{i=1}^{D} \frac{\beta\sigma_i^2}{\alpha+\beta\sigma_i^2}$, where σ_i are singular values of F; (3) maximize α, β by solving $\frac{\partial \mathcal{L}}{\partial \alpha} = 0, \frac{\partial \mathcal{L}}{\partial \beta} = 0$ with m, γ fixed, which yields $\alpha \leftarrow \frac{\gamma}{m^T m}, \beta \leftarrow \frac{n-\gamma}{||Fm-y||_2^2}$. The algorithm is called MacKay's algorithm (Algorithm 2). Section 4.1 gives a theoretical convergence guarantee for the algorithm. Interestingly, the fixed point iteration used for the convergence analysis can also be used in practice to obtain a new and faster algorithm for evidence maximization (Algorithm 3). Please refer to Section 4.1 for details.

After the convergence of evidence maximization, the logarithm maximum evidence $\mathcal{L}(\alpha^*, \beta^*)$ is used to evaluate the compatibility between features and labels. Because $\mathcal{L}(\alpha^*, \beta^*)$ scales linearly with n, we normalize it as $\frac{\mathcal{L}(\alpha^*,\beta^*)}{n}$ and term it **LogME** (logarithm of maximum evidence). Discussion on the influence of dimensionality D is presented in Section 4.2. LogME can be intuitively interpreted as the logarithm of maximum label evidence given pre-trained features.

Extending LogME to complex cases. The LogME approach starts from a single-target regression. If the target problem is a multivariate regression task, *i.e.*, $Y \in \mathbb{R}^{n\times C}$, we can calculate LogME for each dimension c ($1 \le c \le C$) and average them over the C dimensions. If the target problem is a classification task with C classes, Equation 1 cannot be calculated analytically (Daunizeau, 2017) with a categorical prior distribution. State-of-the-art approximation methods like Laplace approximation (Immer et al., 2021) work well in toy data, but perform unsatisfyingly in realistic tasks, as mentioned later in Section 6.1. Therefore, we turn to an alternative solution: convert the classification labels to one-hot labels and treat the problem as multivariate regression. This approach also works for multi-label classification. This way, LogME can be used in both (single-label and multi-label) classification and regression tasks.

The overall algorithmic specification of LogME is presented in Algorithm 1.

3.2.3 Computational speedup

Although the Bayesian approach of maximum evidence has a rigorous theoretical explanation (Knuth et al., 2015), it inherits the drawback of Bayesian methods with respect to high computational complexity. A naïve implementation of Algorithm 2 results in a total complexity of $\mathcal{O}(CD^3 + nCD^2)$. For typical usage with $D \approx 10^3, n \approx 10^4, C \approx 10^3$, the computational cost is 10^{13}, with the wall-clock time comparable to fine-tuning PTM ϕ.

Our conference paper (You et al., 2021) accelerated the computation by avoiding matrix inversion and matrix-matrix multiplication, as shown in Line 8 of Algorithm 2. In this paper, we present a convergence analysis of MacKay's algorithm by fixed point iteration. It turns out that the analysis implies a faster algorithm for evidence maximization. The algorithm is presented in Algorithm 3 and its rationale is explained in Section 4.1.

Algorithm 1 LogME

Input: Pre-trained model ϕ and target dataset $\mathcal{D} = \{(x_i, Y_i)\}_{i=1}^n$
Output: Logarithm of Maximum Evidence (LogME)
Extract features using pre-trained model ϕ: $F \in \mathbb{R}^{n \times D}$, $f_i = \phi(x_i)$, $Y \in \mathbb{R}^{n \times C}$
Compute SVD of F: $F = U\Sigma V^T$. Then $F^T F = V \text{diag}\{\sigma^2\} V^T$
for dimension $c = 1$ to C **do**
    Let $y = Y^{(c)} \in \mathbb{R}^n$,
    Calculate the LogME value $\mathcal{L}_c$ by evidence maximization (Algorithm 2 or Algorithm 3).
end for
Return LogME $\frac{1}{C} \sum_{c=1}^{C} \mathcal{L}_c$

Algorithm 2 Evidence Maximization by MacKay's Algorithm

Input: Extracted features $F \in \mathbb{R}^{n \times D}$ and corresponding labels $y \in \mathbb{R}^n$
Output: Logarithm of Maximum Evidence (LogME)
Note: F has been pre-decomposed into $F = U\Sigma V^T$
Initialize $\alpha = 1, \beta = 1$
while α, β not converge **do**
    Compute $\gamma = \sum_{i=1}^{D} \frac{\beta \sigma_i^2}{\alpha + \beta \sigma_i^2}, \Lambda = \text{diag}\{(\alpha + \beta \sigma^2)\}$
    Naïve: $A = \alpha I + \beta F^T F, m = \beta A^{-1} F^T y$
    Optimized by You et al. (2021): $m = \beta(V(\Lambda^{-1}(V^T(F^T y))))$
    Update $\alpha \leftarrow \frac{\gamma}{m^T m}, \beta \leftarrow \frac{n - \gamma}{||Fm - y||_2^2}$
end while
Compute and return $\mathcal{L} = \frac{1}{n}\mathcal{L}(\alpha, \beta)$ using Equation 2

Algorithm 3 Evidence Maximization by Optimized Fixed Point Iteration

Input: Extracted features $F \in \mathbb{R}^{n \times D}$ and corresponding labels $y \in \mathbb{R}^n$
Output: Logarithm of Maximum Evidence (LogME)
Require: Truncated SVD of F: $F = U_r \Sigma_r V_r^T$, with $U_r \in \mathbb{R}^{n \times r}, \Sigma_r \in \mathbb{R}^{r \times r}, V_r \in \mathbb{R}^{D \times r}$.
Compute the first r entries of $z = U_r^T y$
Compute the sum of remaining entries $\Delta = \sum_{i=r+1}^{n} z_i^2 = \sum_{i=1}^{n} y_i^2 - \sum_{i=1}^{r} z_i^2$
Initialize $\alpha = 1, \beta = 1, t = \frac{\alpha}{\beta} = 1$
while t not converge **do**
    Compute $m^T m = \sum_{i=1}^{r} \frac{\sigma_i^2 z_i^2}{(t + \sigma_i^2)^2}$, $\gamma = \sum_{i=1}^{r} \frac{\sigma_i^2}{t + \sigma_i^2}$, $||Fm - y||_2^2 = \sum_{i=1}^{r} \frac{z_i^2}{(1 + \sigma_i^2/t)^2} + \Delta$
    Update $\alpha \leftarrow \frac{\gamma}{m^T m}, \beta \leftarrow \frac{n - \gamma}{||Fm - y||_2^2}, t = \frac{\alpha}{\beta}$
end while
Compute $m = V_r \Sigma' z$, where $\Sigma'_{ii} = \frac{\sigma_i}{t + \sigma_i^2} (1 \le i \le r)$.
Compute and return $\mathcal{L} = \frac{1}{n}\mathcal{L}(\alpha, \beta)$ using Equation 2

Table 2: The complexity of Algorithm 1 with three implementations of evidence maximization. n, C are the number of samples and the number of classes in classification (or the number of target variables in regression) in downstream tasks, and D is the dimension of features produced by a pre-trained model.

Evidence maximization method	Complexity per while-loop	Overall complexity
naïve implementation	$\mathcal{O}(D^3 + nD^2)$	$\mathcal{O}(nCD^2 + CD^3)$
optimized by You et al. (2021)	$\mathcal{O}(D^2 + nD)$	$\mathcal{O}(nD^2 + nCD + CD^2 + D^3)$
fixed point iteration (this paper)	$\mathcal{O}(n)$	$\mathcal{O}(nD^2 + nCD)$

Table 2 compares the complexity of calculating LogME with three implementations of evidence maximization. The naïve implementation is biquadratic, You et al. (2021) made it cubic, and this paper further reduces the number of cubic terms. The optimized algorithm makes a time-consuming Bayesian approach fast enough, reducing the wall-clock time by order of 10^2 (see Section 6.5 for a quantitative measurement). Note that three implementation methods are functionally equivalent and only differ in computational complexity. Therefore, *the fixed point iteration proposed in this paper is used by default in our implementation.*

4. Theoretical Analyses of LogME

In this section, we analyze two theoretical aspects of the proposed LogME, which further explains the rationale behind the LogME algorithm and helps provide insight into why LogME works.

4.1 Convergence analysis of evidence maximization

Historical remarks: The evidence maximization procedure in Section 3.2.2 was proposed by MacKay (1992) as a heuristic method to maximize the evidence of given data, following the spirit of empirical Bayesian learning (Bishop, 1995). Theoretical analysis is missing and it has been employed as a heuristic in modern machine learning practice. Progress was made in the theoretical justification by Li et al. (2016) who noted that if the predictive uncertainty β is known, the maximization over model uncertainty α can be viewed as a special instantiation of the EM algorithm (Dempster et al., 1977). However, pre-determining β is suboptimal, and in practice α, β are simultaneously maximized. In this paper we provide an analysis of MacKay's algorithm in which α, β are jointly optimized.

We collect necessary notation here: n is the number of data examples; D is the size of feature dimensionality; $F \in \mathbb{R}^{n \times D}$ is the feature matrix, with $r = \text{rank}(F)$ being its rank; $y \in \mathbb{R}^n$ is the label vector of data examples. We have that $r \leq \min\{n, D\}$.

The key in our analysis is to take full advantage of the singular value decomposition of the feature matrix $F = U\Sigma V^T$, where $U \in \mathbb{R}^{n \times n}$, $V \in \mathbb{R}^{D \times D}$, and $\Sigma \in \mathbb{R}^{n \times D}$. Note that Σ only has r non-zero entries: $\Sigma_{ii} = \sigma_i > 0$ $(1 \leq i \leq r)$ where σ_i^2 is the i-th largest eigenvalue of $F^T F$ and $\sigma_i = 0$ $(r + 1 \leq i \leq \max(n, D))$. To simplify the expression, let $z = U^T y$ be the transformed y under orthogonal bases U, *i.e.*, $y = Uz$.

MacKay's algorithm (Algorithm 2) consists of a while-loop which is presented in Algorithm 4. The key to analyzing the whole algorithm is to analyze each iteration of the while-loop. During each iteration, new values α', β' are computed based on old values α, β, which can be regarded as evaluating a vector-valued function $(\alpha', \beta') = g(\alpha, \beta)$.

Algorithm 4 One iteration of evidence maximization in Algorithm 2.

1: Input: α, β; Output: α', β' for the next iteration.
2: Compute $A = \alpha I + \beta F^T F, m = \beta A^{-1} F^T y, \gamma = \sum_{i=1}^{D} \frac{\beta \sigma_i^2}{\alpha + \beta \sigma_i^2}$
3: Return $\alpha' = \frac{\gamma}{m^T m}, \beta' = \frac{n-\gamma}{||Fm-y||_2^2}$

MacKay's algorithm converges if and only if $(\alpha', \beta') = (\alpha, \beta)$ in Algorithm 4. With F, y as constants, the convergence of Algorithm 2 is equivalent to the existence of the fixed point of the vector-valued function g, *i.e.*, the existence of α, β such that $(\alpha, \beta) = g(\alpha, \beta)$.

In general, fixed points of vector-valued functions are difficult to analyze and visualize. Fortunately, we find that the vector-valued function $(\alpha', \beta') = g(\alpha, \beta)$ is homogeneous: $g(k\alpha, k\beta) = kg(\alpha, \beta), \forall k > 0$. Let $t = \alpha/\beta$, and $t' = \alpha'/\beta'$, the vector-valued function $(\alpha', \beta') = g(\alpha, \beta)$ induces a scalar function $t' = f(t)$, whose explicit form can be derived in Theorem 1. Evaluating $g(\alpha, \beta)$ is equivalent to calculating $f(\frac{\alpha}{\beta})$, which is easier to analyze.

Theorem 1 Algorithm 4 induces a scalar function (Equation 3) with $t = \frac{\alpha}{\beta}$ and $t' = \frac{\alpha'}{\beta'}$.

$$t' = f(t) = \left(\frac{n}{n - \sum_{i=1}^{D} \frac{\sigma_i^2}{t + \sigma_i^2}} - 1 \right) t^2 \frac{\sum_{i=1}^{n} \frac{z_i^2}{(t+\sigma_i^2)^2}}{\sum_{i=1}^{n} \frac{\sigma_i^2 z_i^2}{(t+\sigma_i^2)^2}}. \tag{3}$$

The proof is in Appendix B. Although $f(t)$ seems very complicated and completely understanding its behavior is difficult, surprisingly, the existence of a fixed point of $f(t)$ can be guaranteed with an interpretable condition, as presented in the following Theorem 2.

Theorem 2 If $r < n$ and $\sum_{1 \le i,j \le n} (z_i^2 - z_j^2)(\sigma_i^2 - \sigma_j^2) > 0$, then $f(t)$ has a fixed point and thus MacKay's algorithm will converge.

The proof is in Appendix C. Theorem 2 requires two conditions to guarantee the fixed point: $r < n$ and $\sum_{1 \le i,j \le n} (z_i^2 - z_j^2)(\sigma_i^2 - \sigma_j^2) > 0$. The first condition is easy to interpret and can be easily satisfied: usually $n > D$, and $n > D \ge r$ naturally holds. The condition $\sum_{1 \le i,j \le n} (z_i^2 - z_j^2)(\sigma_i^2 - \sigma_j^2) > 0$ is new in this paper. Note that $z = U^T y$ and $z_i = U_i^T y$, where U_i (the i-th column of U) is the left-singular vector of the singular value σ_i, which means that z_i is the projection of label vector y in the direction of the left-singular vector for the singular value σ_i. Intuitively speaking, $\sum_{1 \le i,j \le n} (z_i^2 - z_j^2)(\sigma_i^2 - \sigma_j^2) > 0$ requires z_i^2 to share roughly the same descending order as σ_i^2. For larger σ_i^2 (*i.e.*, smaller i), it means the projection of y in the corresponding left-singular vector should be larger, which can be interpreted as a rigorous way to say that *labels y are meaningful with respect to the features F*. We would like to emphasize that the requirement on the order of z_i^2 is *soft*: strict order $z_i^2 \ge z_j^2 \iff i \le j \iff \sigma_i^2 \ge \sigma_j^2$ certainly assures the convergence condition

$\sum_{1\leq i,j\leq n}(z_i^2-z_j^2)(\sigma_i^2-\sigma_j^2)>0$, but as long as most z_i^2 follow the order, the condition can be satisfied. We find that all experiments in this paper admit the convergence condition, *i.e.*, the evidence maximization algorithm is guaranteed to converge if the data is meaningful. For example, Figure 4 plots $f(t)$ on the CIFAR10 dataset, which clearly shows cross points of $f(t)$ and t, so the convergence condition $\sum_{1\leq i,j\leq n}(z_i^2-z_j^2)(\sigma_i^2-\sigma_j^2)>0$ holds.

Make the fixed point iteration faster. Note that the fixed point iteration Equation 3 requires explicitly computing $z=U^Ty$ with $\mathcal{O}(n^2)$ storage and computation, which would be undesirable if n is very large. To obtain a practical algorithm, we take advantage of the fact that $\sigma_i=0$ for $i>r$, and optimize the fixed point iteration as follows:

$$
\begin{aligned}
t'=f(t)&=\left(\frac{n}{n-\sum_{i=1}^{D}\frac{\sigma_i^2}{t+\sigma_i^2}}-1\right)t^2\frac{\sum_{i=1}^{n}\frac{z_i^2}{(t+\sigma_i^2)^2}}{\sum_{i=1}^{n}\frac{\sigma_i^2z_i^2}{(t+\sigma_i^2)^2}}\\
&=\left(\frac{n}{n-\sum_{i=1}^{r}\frac{\sigma_i^2}{t+\sigma_i^2}}-1\right)t^2\frac{\sum_{i=1}^{r}\frac{z_i^2}{(t+\sigma_i^2)^2}+\frac{1}{t^2}\sum_{i=r+1}^{n}z_i^2}{\sum_{i=1}^{r}\frac{\sigma_i^2z_i^2}{(t+\sigma_i^2)^2}}\\
&=\left(\frac{n}{n-\sum_{i=1}^{r}\frac{\sigma_i^2}{t+\sigma_i^2}}-1\right)t^2\frac{\sum_{i=1}^{r}\frac{z_i^2}{(t+\sigma_i^2)^2}+\frac{1}{t^2}(\sum_{i=1}^{n}y_i^2-\sum_{i=1}^{r}z_i^2)}{\sum_{i=1}^{r}\frac{\sigma_i^2z_i^2}{(t+\sigma_i^2)^2}}.
\end{aligned}
$$

Therefore, we can derive a faster algorithm (Equation 4) for the fixed point iteration, which only requires the first r entries of z without computing the full U matrix or the full z vector. This is the algorithm we implement in Algorithm 3, setting:

$$
t'=f(t)=\left(\frac{n}{n-\sum_{i=1}^{r}\frac{\sigma_i^2}{t+\sigma_i^2}}-1\right)t^2\frac{\sum_{i=1}^{r}\frac{z_i^2}{(t+\sigma_i^2)^2}+\frac{1}{t^2}(\sum_{i=1}^{n}y_i^2-\sum_{i=1}^{r}z_i^2)}{\sum_{i=1}^{r}\frac{\sigma_i^2z_i^2}{(t+\sigma_i^2)^2}}. \tag{4}
$$

4.2 Influence of dimensionality

In Section 3.2.2, we normalize the LogME value by the number of examples because Equation 2 scales linearly with n. The influence of feature dimension D is, however, unclear. In this section, we find two cases (feature duplicate and feature padding) where *LogME value remains unchanged when the feature dimension goes up without introducing more information.* The two cases show the existence of infinitely many features with arbitrary feature dimensions that share the same LogME value, therefore removing the necessity of dimensionality normalization.

Corollary 3 (feature duplicate) LogME value will remain the same if the feature consists of arbitrary replicas of the original feature. Formally speaking, if the LogME value for $F\in\mathbb{R}^{n\times D}$ and $y\in\mathbb{R}^n$ is $\mathcal{L}$, then the LogME value for $\tilde{F}=[F,...,F]\in R^{n\times qD}$ and $y\in\mathbb{R}^n$ is also $\mathcal{L}$. ($q\in\mathbb{N}$ is a natural number to represent the number of replicas.)

Corollary 4 (feature padding) LogME value will remain the same if the feature is padded with an arbitrary number of zeros. Formally speaking, if the LogME value for $F\in\mathbb{R}^{n\times D}$ and $y\in\mathbb{R}^n$ is $\mathcal{L}$, then the LogME value for $\tilde{F}=[F,\mathbf{0}]\in R^{n\times(D+d)}$ and $y\in\mathbb{R}^n$ is also $\mathcal{L}$. ($d\in\mathbb{N}$ is a natural number and $\mathbf{0}\in\mathbb{R}^{n\times d}$ is a matrix with all zero entries.)

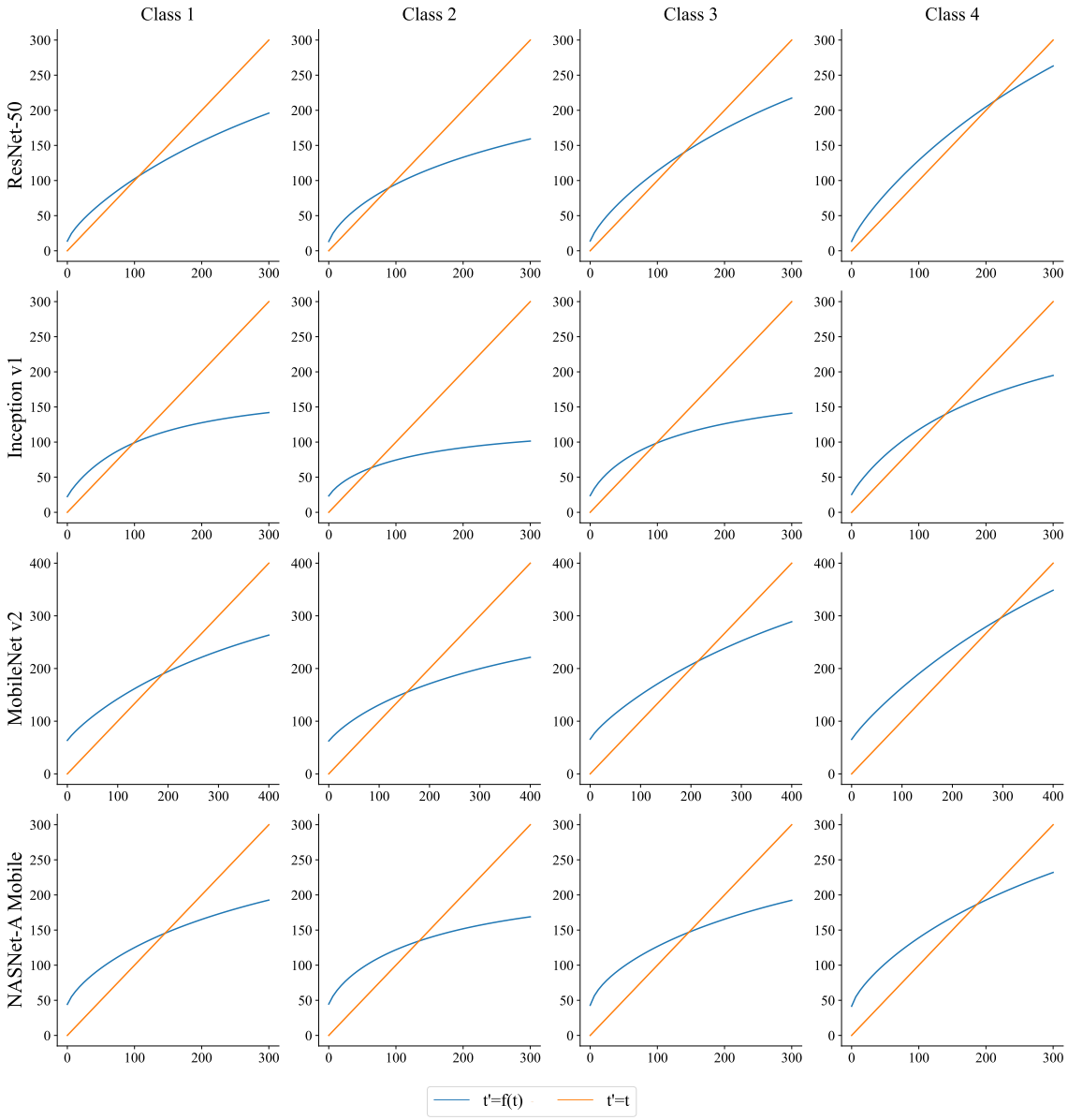


Figure 4: Fixed points of $f(t)$ in Equation 3 for the first four classes in CIFAR10 with four pre-trained models. The full figure for all the ten classes in CIFAR10 with five pre-trained models can be found in Figure 13, which is omitted here to prettify the layout. We plot $t' = f(t)$ (in blue) and $t' = t$ (in orange), whose intersections are fixed points $f(t) = t$. The existence of fixed points guarantees the convergence of MacKay's algorithm for evidence maximization.

The proofs of Corollary 3 and Corollary 4 are in Appendix D and Appendix E, respectively. The core idea is to find the closed-form relationship between decompositions of $\tilde{F}$ and F.

Corollary 3 and Corollary 4 imply that duplicating features or padding features with zeros will not change the value of LogME. LogME is capable of filtering out redundant information in features, explaining its excellent empirical performance in You et al. (2021).

5. Tuning Pre-Trained Models

The new paradigm we propose consists of ranking and tuning pre-trained models. Sections 3 and 4 described technical background for ranking pre-trained models, including the transferability metric LogME and its theoretical analyses. In this section we focus on tuning pre-trained models, completing the overall paradigm.

We identify two possible scenarios in the tuning of pre-trained models: single best PTM tuning and multiple PTM tuning. (1) *Single best PTM tuning* is suitable if there are no constraints on the network architecture, parameter count, or FLOPs of computation. These constraints are common in industrial applications, but are less important in academic research. Therefore, single best PTM tuning is common in academic research. Intuitively, the remaining PTMs are considered inferior to the best ranked PTM so they are not considered worth the effort to identify and deploy. We refer readers to dedicated papers (Chen et al., 2019; Kou et al., 2020; You et al., 2020) on how to fine-tune a single PTM. (2) When we deploy neural networks in industrial applications, typically there are strict constraints on the budget of memory footprint or power consumption. Therefore, the pre-trained model ϕ_t satisfying these constraints is probably not the best ranked. The current state of the art is that practitioners can only fine-tune ϕ_t, and the overall knowledge of the PTM hub $\{\phi_i\}_{i=1}^M$ cannot be exploited. In this paper, we show that it is possible to transfer knowledge from several teacher PTMs $\{\phi_k\}_{k=1}^K$ to the target pre-trained model ϕ_t during fine-tuning, a paradigm which we call "multiple PTM tuning."

A side issue in tuning multiple PTMs is how to select the teacher PTMs. Typically, $K < M$, *i.e.*, not all PTMs are necessary, since some PTMs may not be suitable for the target task and would hinder the transfer learning procedure. However, for M pre-trained models, the possible number of teacher combinations is $\mathcal{O}(2^M)$, which is impractical to enumerate. To overcome the exponential complexity, we avail ourselves of the PTM ranking. With the PTM ranking, we can greedily select teacher PTMs according to the rank. For example, if we want to choose K teacher PTMs, then the top-K ranked PTMs $\{\phi_k\}_{k=1}^K$ are the teacher for knowledge transfer. As for how to choose the hyper-parameter K ($1 \leq K \leq M$), we give empirical guidelines in Section 6.3.

Multiple PTM tuning offers a unique advantage over simply fine-tuning the target pre-trained model ϕ_t: if the specified target pre-trained model ϕ_t is not the best-ranked, we can still improve it by transferring knowledge from top-performing PTMs $\{\phi_k\}_{k=1}^K$.

5.1 Problem setup for multiple PTM tuning

Now suppose we have selected K pre-trained models $\{\phi_k\}_{k=1}^K$, with each pre-trained model ϕ_k transforming input x into a D_k-dimensional feature vector. In general, pre-trained models $\{\phi_k\}_{k=1}^K$ have various network architectures, and dimensionality of features $\{D_k\}_{k=1}^K$ can vary. Let ϕ_t be the target architecture, which transforms the input into D_t dimensional

feature vector. Formally, the multiple PTM tuning problem is to fine-tune a pre-trained model ϕ_t by leveraging selected pre-trained models $\{\phi_k\}_{k=1}^K$, as shown in Figure 2.

To fine-tune a model ϕ_t in a target task, a new output head would be attached after ϕ_t, where a target-specific loss is calculated. The target-specific head and loss are necessary for every possible solution to multiple PTM tuning, which is taken care of by a loss function L_{task}. We will not elaborate on L_{task} as it varies with tasks. Next, in Section 5.2, we summarize existing approaches to the problem and introduce our method in Section 5.3.

5.2 Existing approaches to the problem of multiple PTM tuning

A baseline approach is to fine-tune ϕ_t without considering teacher PTMs $\{\phi_k\}_{k=1}^K$. This can serve as a baseline to measure the improvement brought by multiple PTM tuning.

The knowledge distillation approach to multiple PTM tuning is knowledge distillation (Hinton et al., 2015) in the feature space via mean-square error. Since the feature dimensions may differ between ϕ_t and $\{\phi_k\}_{k=1}^K$, a transformation module is necessary. The knowledge distillation (KD) method takes advantage of selected pre-trained models by adding a regularization term. $L_{KD} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{K}\sum_{k=1}^{K}||\phi_k(x_i) - W_k\phi_t(x_i)||_2^2$, where W_k is learnable parameter to transform a D_t-dimensional feature $\phi_t(x_i)$ into a D_k-dimensional vector compatible with $\phi_k(x_i)$. Even if $D_k = D_t$, the semantics of each dimension in ϕ_t and ϕ_k may vary, making it necessary to introduce the transformation parameter W_k. The final loss is $L_{task} + \lambda L_{KD}$, with hyper-parameter λ trading the two terms. The KD method is another simple but general baseline in multiple PTM tuning. It can be applied to various PTMs but the performance improvement is limited.

Zoo-tuning for homogeneous PTMs tuning. In the special case when ϕ_t and $\{\phi_k\}_{k=1}^K$ all share the same network architecture, Zoo-tuning proposed by Shu et al. (2021) adaptively aggregates parameters of $\{\phi_k\}_{k=1}^K$ into ϕ_t in a layer-wise fashion. It does not modify the loss L_{task}, but changes the training process by model aggregation. Zoo-tuning is the current state-of-the-art method for homogeneous PTM tuning, but it fails to deal with the heterogeneous scenario when architectures of ϕ_t and $\{\phi_k\}_{k=1}^K$ are different.

5.3 B-Tuning: A Bayesian approach to multiple PTM tuning

We draw lessons from the shortcomings of the aforementioned knowledge distillation approach and the Zoo-tuning approach. Knowledge distillation operates at the level of output features, which works for heterogeneous PTMs but aligning features across PTMs is not easy. Zoo-tuning operates at the level of parameters (layers), thereby limiting itself to the homogeneous case. Taking the positive aspects of both frameworks, we design our approach to operate at the level of features to hide the heterogeneity among PTMs, and we go beyond features to avoid explicitly aligning features from various pre-trained models. Inspired by the ranking metric (LogME), we propose an approach that builds on posterior predictive distributions from Bayesian regression.

A posterior predictive distribution is $p(y'|f, F, y) = \int_w p(y'|w, f)p(w|F, y)\mathrm{d}w$, which predicts the label y' of incoming feature f conditioned on all the available training features F and labels y rather than just using f. With pre-computed α^*, β^*, m (byproducts of the LogME algorithm), $p(y'|w, f) \sim \mathcal{N}(w^T f, \beta^{*-1})$ by definition, and $p(w|F, y) = \frac{p(w)p(F,y|w)}{\int_{w'} p(w')p(F,y|w')\mathrm{d}w'}$

by Bayes' theorem. Rasmussen (2003) shows that $p(w|F,y) \sim \mathcal{N}(\beta^* A^{-1} F^T y, A^{-1})$ with $A = \alpha^* I + \beta^* F^T F$. Plugging in the distributions of $p(y'|w,f)$ and $p(w|F,y)$, Rasmussen (2003) shows that $p(y'|f,F,y) \sim \mathcal{N}(f^T m, f^T A^{-1} f + \beta^{*-1})$ with $m = \beta^* A^{-1} F^T y$. In short, for extracted features $F \in \mathbb{R}^{n \times D}$ and labels $y \in \mathbb{R}^n$, the LogME algorithm gives α^*, β^*, m, and the posterior predictive distribution is $p(y'|f,F,y) \sim \mathcal{N}(f^T m, f^T A^{-1} f + \beta^{*-1})$. For a full derivation of the posterior predictive distribution please see Rasmussen (2003).

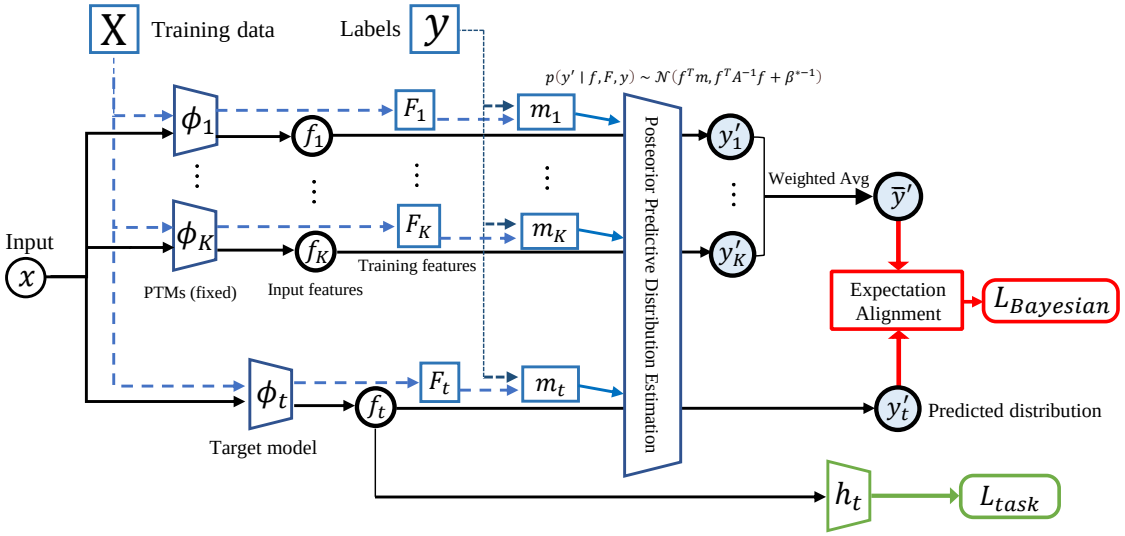


Figure 5: Illustration of B-Tuning. Dashed lines are pre-calculated before tuning.

The posterior predictive distribution depends on the training data (F, y) and input features f. Let F_k be features extracted by the pre-trained model ϕ_k, $f_k = \phi_k(x)$ be the output features of the current data point extracted by the pre-trained model ϕ_k, then each pre-trained model can produce a posterior predictive distribution $p(y'_k|f_k, F_k, y) \sim \mathcal{N}(f_k^T m_k, f_k^T A_k^{-1} f_k + \beta_k^{*-1})$. How to combine these distributions? We propose to mix them according to their LogME values $\{\mathcal{L}_k\}_{k=1}^K$ as a mixture of Gaussians, $\bar{y}' = \sum_{k=1}^K \pi_k y'_k$ where $\pi_k = \frac{\exp(\mathcal{L}_k/t)}{\sum_{j=1}^K \exp(\mathcal{L}_j/t)}$, where t is a temperature hyper-parameter (Hinton et al., 2015) that can be adjusted according to the difference of LogME values. Although $\{y'_k\}_{k=1}^K$ admit simple Gaussian distributions, the exact distribution of mixed $\bar{y}'$ is intractable because the features F_k come from the same dataset and $\{y'_k\}_{k=1}^K$ are dependent. Nevertheless, according to the linearity property of expectation, $\mathbb{E}\bar{y}' = \sum_{k=1}^K \pi_k \mathbb{E} y'_k = \sum_{k=1}^K \pi_k f_k^T m_k$, *the expectation of* $\bar{y}'$ *is known.*

For the target model ϕ_t, the posterior predictive distribution is defined as $p(y'_t|f_t, F_t, y) \sim \mathcal{N}(f_t^T m_t, f_t^T A_t^{-1} f_t + \beta_t^{*-1})$. Since $\bar{y}'$ can be regarded as prior knowledge from pre-trained models $\{\phi_k\}_{k=1}^K$, we can *align the expectation of* y'_t *and* $\bar{y}'$ as a regularization term, $L_{Bayesian} = \frac{1}{n} \sum_{i=1}^n ||\mathbb{E}\bar{y}' - \mathbb{E} y'_t||_2^2$. Note that the expectation is taken over the predictive distributions and can be calculated analytically. Extending the formula to multiple classes, the final

expression of the regularization term is

$$L_{Bayesian} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{C}\sum_{c=1}^{C}(\sum_{k=1}^{K}\pi_k f_k^T m_{k,c} - f_t^T m_{t,c})^2, \tag{5}$$

where $m_{k,c}, m_{t,c}$ are calculated by the LogME algorithm and are fixed during training. The final loss is $L_{task} + \lambda L_{Bayesian}$, with λ introduced to trade off two terms. Because the method depends on the Bayesian approach of calculating posterior predictive distribution, we call it Bayesian Tuning, or **B-Tuning**. Figure 5 describes the method and the computation graph. Only ϕ_t is updated during B-Tuning while the teacher PTMs $\{\phi_k\}_{k=1}^K$ are fixed.

B-Tuning has two advantages over previous methods: (1) It hides the heterogeneity among PTMs by operating at the level of features, yielding a *general* solution to multiple PTM tuning for both the homogeneous and heterogeneous cases. (2) B-Tuning has a simple interpretation: it aligns features adaptively with m serving as an attention-like mechanism, removing the necessity of learning to transform features into a shared space as in the knowledge distillation approach. The superiority of B-Tuning in multiple PTM tuning is empirically demonstrated in Section 6.3.

6. Experiments

This section presents comprehensive experiments. Section 6.1 illustrates the behavior of LogME on toy problems. Experiments on ranking PTMs and tuning PTMs are in Section 6.2 and Section 6.3 respectively, demonstrating the power of the proposed new paradigm. Section 6.5 quantitatively measures the efficiency of LogME and Section 6.6 compares LogME against a common approach of re-training the head over a fixed feature extractor, providing a comprehensive understanding of LogME. Original data for some figures are available in the appendix. Code for LogME is available at `https://github.com/thuml/LogME`.

6.1 Illustration with toy data

To give an intuitive sense of how LogME works, we generate features with increasing noise to mimic the features extracted by pre-trained models with decreasing transferability and to check if LogME can measure the quality of features.

For classification (Figure 6 left), three clusters in a 2-D plane are generated, with colors representing the categories. Initially, the features are well separated so LogME has a large value. Then we add Gaussian noise with increasing variance to the data and the clustering structure in feature space disappears, leading to smaller LogME values as expected.

For regression (Figure 6 right), x is uniformly distributed ($x \sim \mathcal{U}[0, 1]$) and the output $y = 2x + \epsilon$ with observation error $\epsilon \sim \mathcal{N}(0, 0.1^2)$. By adding noise to the feature $x' = x + \mathcal{N}(0, t^2)$, the quality of feature x' becomes worse and it is harder to predict y from x'. With larger t (the standard deviation of noise), LogME becomes smaller as expected.

These toy experiments on synthesized data show that LogME is an effective measure of the feature quality, and therefore can provide a ranking for PTMs in a pre-trained model hub.

Figure 6 also shows the LogME value calculated by Immer et al. (2021) using Laplace approximation. In this toy experiment, both LogME and Laplace approximation correctly

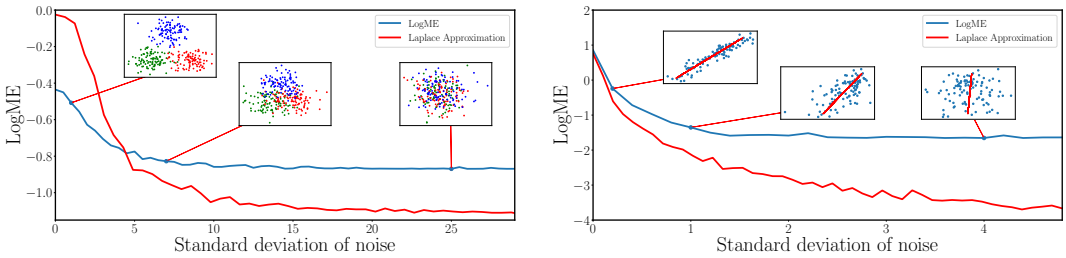


Figure 6: Experiments with toy data demonstrate that LogME values go down with decreasing feature quality. "Laplace Approximation" means that the LogME value calculated by Immer et al. (2021) use the Laplace approximation.

measure the trend of feature quality. In regression, the Laplace Approximation is strictly lower than LogME; in classification, Laplace approximation uses a categorical prior and approximates the marginal likelihood, while LogME converts classification labels to one-hot labels (with a Gaussian prior) and calculates the exact value without approximation. The left plot in Figure 6 confirms that both approaches can reflect the trend of feature quality. However, we notice that Laplace approximation has larger fluctuations than LogME, and the Laplace approximation requires more computation than LogME. In addition, its performance in realistic data (Section 6.2.1) is not satisfactory. Therefore, when dealing with classification data, we convert classification labels to one-hot labels and treat the problem as a multivariate regression problem in LogME. How to analytically calculate the value with a categorical prior is left as a future research question.

6.2 Ranking pre-trained models

This section focuses on the first part of the proposed paradigm: ranking pre-trained models. The goal is to rank pre-trained models so that potentially best PTMs can be selected for the subsequent tuning process. This section attaches great importance to the diversity of pre-trained models and downstream tasks. Section 6.2.1 and Section 6.2.2 transfer supervised pre-trained models to classification and regression tasks, respectively; Section 6.2.3 explores unsupervised pre-trained models on both classification and regression; Section 6.2.4 and Section 6.2.5 study pre-trained language models on language understanding tasks and a sequential tagging task, respectively. These extensive experiments demonstrate the generality and effectiveness of the proposed LogME method in ranking pre-trained models.

6.2.1 Ranking supervised pre-trained models in classification tasks

We use 12 ImageNet pre-trained models available from PyTorch: Inception V1 (Szegedy et al., 2015), Inception V3 (Szegedy et al., 2016), ResNet 34 (He et al., 2016), ResNet 50 (He et al., 2016), ResNet 101 (He et al., 2016), ResNet 152 (He et al., 2016), Wide ResNet 50 (Zagoruyko and Komodakis, 2016), DenseNet 121 (Huang et al., 2017), DenseNet 169 (Huang et al., 2017), DenseNet 201 (Huang et al., 2017), MobileNet V2 (Sandler et al., 2018), and NASNet-A Mobile (Tan et al., 2019). These pre-trained models cover most of the supervised pre-trained models in transfer learning that practitioners frequently use.

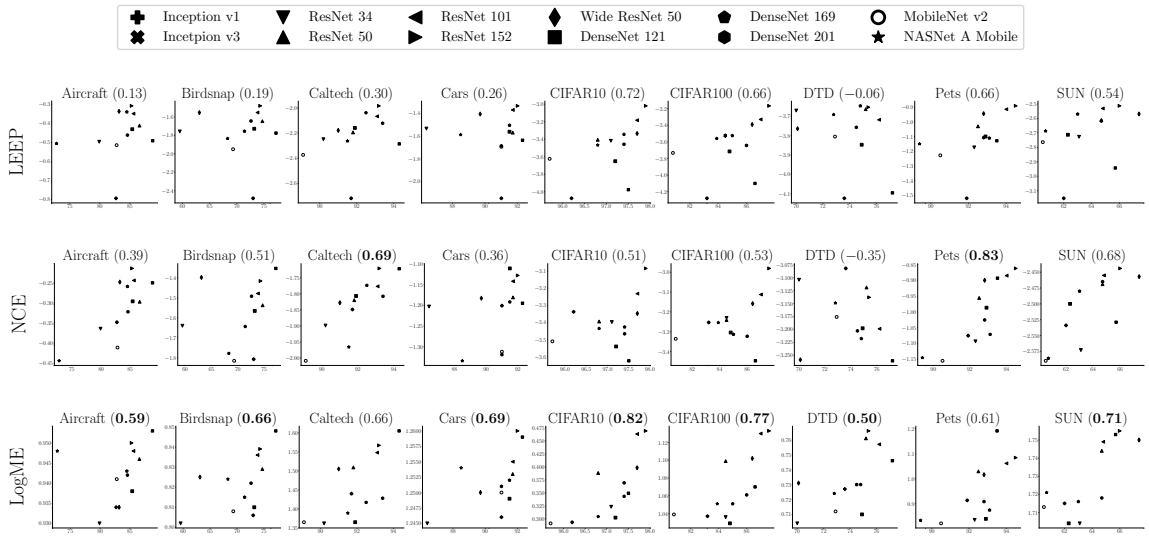


Figure 7: Correlation values (τ_w) between fine-tuned accuracy (X-axis) and scores produced by three methods (Y-axis) for ranking PTMs on 9 datasets with 12 pre-trained models. One row for each method, one column for each dataset (with τ_w in the parenthesis near the dataset name), and one marker for each pre-trained model. The best τ_w in each dataset is marked in bold.

For downstream classification tasks, we take nine commonly used datasets: Aircraft (Maji et al., 2013), Birdsnap (Berg et al., 2014), Caltech (Fei-Fei et al., 2004), Cars (Krause et al., 2013), CIFAR10 (Krizhevsky and Hinton, 2009), CIFAR100 (Krizhevsky and Hinton, 2009), DTD (Cimpoi et al., 2014), Pets (Parkhi et al., 2012), and SUN (Xiao et al., 2010). The description of each dataset and data statistics are listed in Appendix F.

For all the datasets we use, we respect the official train/val/test splits if they exist, otherwise we use 60% data for training, 20% data for validation (searching hyper-parameters to measure the reference transfer learning performance) and 20% data for testing. Models are trained with a fixed number of epochs, and the best model in the validation split is used as the final model to be tested in the test split.

To compute the reference transfer learning performance, $\{T_m\}_{m=1}^M$ ($M = 12$), we carefully fine-tune pre-trained models with grid-search of hyper-parameters. Li et al. (2020) pointed out that learning rate and weight decay are the two most important hyper-parameters. Hence, we grid-search the learning rate and weight decay (seven learning rates from 10^{-1} to 10^{-4}, and seven weight decays from 10^{-6} to 10^{-3}, all logarithmically spaced) to select the best hyper-parameter on the validation split and compute the accuracy on the test split as the reference transfer learning performance. *It is noteworthy that LogME requires neither fine-tuning nor grid search.* Here we fine-tune pre-trained models to see how well the LogME values correlate with the reference transfer performance, but practitioners can straightforwardly use LogME to evaluate pre-trained models without fine-tuning.

We compare LogME against LEEP (Nguyen et al., 2020) and NCE (Tran et al., 2019). Results of calculating the evidence using Laplace approximation (Immer et al., 2021) are

not shown but are listed in the appendix, where we document that its performance is unsatisfactory. Before this paper, LEEP and NCE were the only two methods to rank PTMs without fine-tuning, and they can only rank supervised pre-trained models in classification tasks. We use LEEP, NCE, and LogME to compute scores $\{S_m\}_{m=1}^M$ by applying 12 pre-trained models to the datasets. The correlation values τ_w between scores and fine-tuned accuracies are presented in Figure 7.

We can find that LogME has consistently better correlation than LEEP, and outperforms NCE on most datasets (7 datasets out of 9 datasets). Note that LEEP and NCE even show negative correlation values in DTD (Cimpoi et al., 2014), because they rely on the relationship between classes of the pre-trained task and the target task but DTD classes (textures) are very different from ImageNet categories (objects). In contrast, LogME still performs reasonably well for DTD.

According to the interpretation of τ_w in Section 3.1, correlation value τ_w can be roughly translated into $\frac{\tau_w+1}{2}$ probability of correct comparison (concordant pairs). The smallest τ_w of LogME in Figure 7 is around 0.5, so the probability of a pre-trained model ϕ_A transferring better than ϕ_B is about 75% if ϕ_A has a larger LogME. For most tasks τ_w of LogME is 0.7 or 0.8, so the probability of correct selection is 85% or 90%, sufficient for practical usage.

6.2.2 Ranking supervised pre-trained models in a regression task

We now turn to an evaluation of how well LogME can assess pre-trained models for a regression task. The two prior methods (LEEP and NCE) depend on a categorical relationship between pre-trained categories and downstream categories, therefore they do not apply to regression tasks.

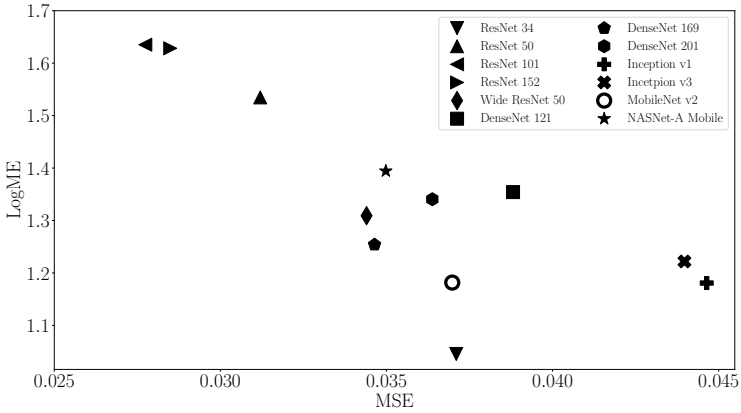


Figure 8: Supervised pre-trained models transferred to dSprites.

The regression task we use is the dSprites (Matthey et al., 2017) dataset from the Visual Task Adaptation Benchmark (Zhai et al., 2020), a common benchmark for evaluating the quality of learned representations. The input is an image containing a sprite (heart, square, and ellipse) with varying scale/orientation/position. Pre-trained models are transferred to predict four scalars (scale, orientation, and (x, y) positions) together, and mean square error (MSE) on the test data is reported. The supervised pre-trained models and the hyper-parameter tuning scheme are the same as in Section 6.2.1.

Results are plotted in Figure 8. It is clear that LogME and MSE are well correlated and the correlation coefficient $\tau_w = 0.79$ is very large: if a pre-trained model ϕ_A has larger LogME than ϕ_B, with roughly 89.5% probability ϕ_A is better (has smaller MSE) than ϕ_B after actually fine-tuning.

6.2.3 Ranking contrastive pre-trained models in downstream tasks

Unsupervised pre-trained models have attracted much attention due to their potential ability to exploit massive unlabeled datasets on the Internet (He et al., 2020). They use a contrastive loss (Gutmann and Hyvärinen, 2010) to inject supervision signals into pre-training with unlabeled data, and they feature a projection head with continuous output. Ranking contrastive pre-trained models is an important emerging challenge, and unfortunately current models such as LEEP and NCE cannot be extended to deal with the projection head of contrastive-based unsupervised pre-trained models because they rely on discrete categorical relationships.

Since LogME only requires features extracted from pre-trained models, it can be applied to contrastive pre-trained models. To demonstrate this, we use four popular models pre-trained with various training schemes: MoCo V1 (He et al., 2020) with momentum contrast, MoCo V2 (Chen et al., 2020b) with an MLP projection head and strong data augmentation, MoCo 800 trained with 800 epochs as suggested by Chen et al. (2020a), and SimCLR (Chen et al., 2020a) trained by a carefully designed training scheme (Chen et al., 2020a).

For classification, we use Aircraft (Maji et al., 2013), the first dataset (alphabetically) in Section 6.2.1; for regression, we use dSprites (Matthey et al., 2017), the only regression task in this paper. Results are shown in Table 3. SimCLR on dSprites is not reported as it does not converge after several trials, possibly because it is heavily tailored to classification tasks. LogME gives a *perfect ordering* of both accuracy and MSE. Note that the reference order on transfer learning performance in Aircraft (MoCo V1 < MoCo V2 < MoCo 800) is different from the order in dSprites (MoCo V1 < MoCo 800 < MoCo V2), emphasizing that ranking pre-trained models is *task adaptive*. We also observe that LogME values of unsupervised pre-trained models are similar (the difference is smaller than their supervised counterparts in Section 6.2.1), mainly because unsupervised features are not very discriminative.

Table 3: Use LogME to rank unsupervised pre-trained models.

PTM	Aircraft		dSprites	
	Accuracy (%)	LogME	MSE	LogME
MoCo V1	81.68	0.934	0.069	1.52
MoCo V2	84.16	0.941	0.047	1.64
MoCo 800	86.99	0.946	0.050	1.58
SimCLR	88.10	0.950	-	-
	τ_w: 1.0		τ_w: 1.0	

6.2.4 Ranking pre-trained language models in the GLUE benchmark

To further demonstrate the generality of LogME, we show how LogME can work for pre-trained language models. Again, existing methods (LEEP and NCE) cannot deal with these pre-trained language models.

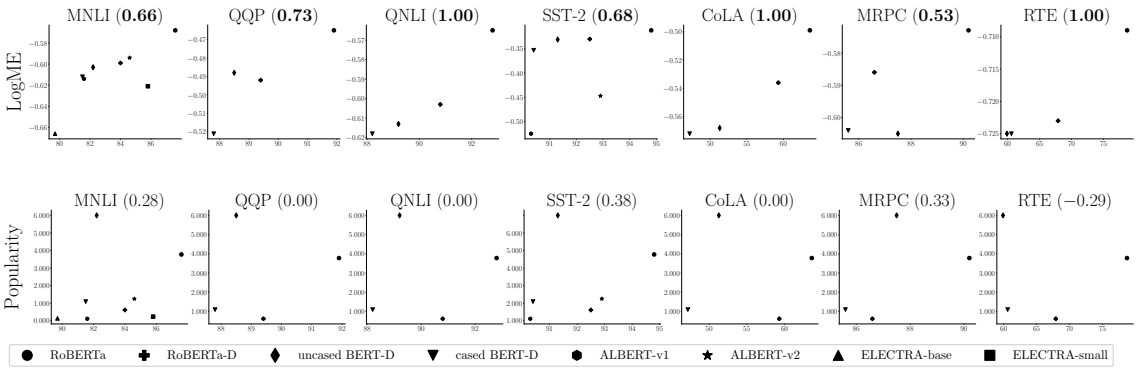


Figure 9: Correlation values (τ_w) between fine-tuned accuracy (X-axis) and LogME value / Popularity value (Y-axis) in seven GLUE tasks with eight popular PTMs. One sub-figure for each task (with its τ_w in the parenthesis), and one marker for each PTM.

Here we take another approach to evaluating the reference transfer performance $\{T_m\}_{m=1}^M$. We do not fine-tune pre-trained models, but directly take fine-tuned results from HuggingFace Models, and check if LogME values can correlate well with the results. Specifically, we take pre-trained models that have GLUE performance tuned by the HuggingFace organization, and select the top eight downloaded models: RoBERTa (Liu et al., 2019), RoBERTa-D, uncased BERT-D, cased BERT-D, ALBERT-v1 (Lan et al., 2020), ALBERT-v2 (Lan et al., 2020), ELECTRA-base (Clark et al., 2020), and ELECTRA-small (Clark et al., 2020) ("D" means distilled version). The LogME values on seven GLUE tasks together with fine-tuned accuracies are plotted in Figure 9. Some models only have results for certain tasks and we keep them as they are. Even though these accuracy numbers are tuned by the HuggingFace organization, LogME perfectly estimates the ranking of transfer performance for three tasks (with $\tau_w = 1$), showing the surprising effectiveness of LogME in ranking pre-trained models.

One may wonder how well a pre-trained model's popularity indicates its transfer learning performance, because it is a common belief that PTMs with consistent improvements across many tasks may tend to become popular. To address this question, a quantitative measurement of popularity is required. We consider two possible quantities: the citation number of the paper proposing the pre-trained model, and the download count of the pre-trained model. The paper citation number is not a proper metric for assessing individual PTM's transferability, because one paper can contain many PTMs. For example, the BERT paper (Devlin et al., 2019) contains BERT-base and BERT-large, which have the same citation number but are in different transferability levels. Download count is a PTM-wise well-defined metric, hence we can use it as a proxy for popularity.

Thanks to the public data from HuggingFace, each PTM's download count (measured in millions) is available to approximate the popularity. The bottom figure in Figure 9 shows

how well popularity performs when it is used as a transferability metric. It is clear that popularity does not correlate well with transfer learning performance: *the τ_w values of popularity are significantly lower than LogME's τ_w values*, and negative correlation values occur in the RTE task. Note that BERT models are the most popular but RoBERTa is the best among these tasks, revealing a mismatch between popularity and transfer learning performance. These experiments provide further justification for the motivation of this paper—practitioners usually select the most popular pre-trained model due to the lack of a satisfying selection strategy, and LogME can come to their rescue.

6.2.5 Ranking pre-trained language models in a sequential tagging task

So far, we have only considered simple classification and regression tasks. It would be valuable to extend LogME to tasks with structured output such as object detection and semantic segmentation. Next we show how LogME can be used in a sequential tagging task where both the input and the output are structured. How to deal with a general task with structured output is left as future work.

The specific task we consider in this section is named entity recognition (Sang and De Meulder, 2003). It requires the model to predict the entity label (person, location, organization, etc.) of every token in a sentence, therefore the output is structured. Considering that the named entity recognition task is sometimes referred to as "token-level classification," we can flatten the token dimension to apply LogME. The only change is that n represents the number of tokens rather than the number of sentences.

We use the same PTMs as in Section 6.2.4, and the dataset is CoNLL-2003 (Sang and De Meulder, 2003) whose performance is measured by F-1 score. Table 4 holds the results. The rank correlation value τ_w is 0.20, smaller than results in previous sections. The small τ_w is caused by an outlier PTM named RoBERTa (Liu et al., 2019), which has the largest F-1 score with a relatively small LogME value. We conjecture that RoBERTa has a small LogME value because it is trained much longer than BERT in the masked language modeling task, which might make its representation tailored to the task, lowering its LogME score in the dissimilar task of named entity recognition. On the other hand, RoBERTa is robustly optimized, so it can be easily fine-tuned to downstream tasks with competitive results.

If we select the best PTM by the largest LogME value, ALBERT-v1 will be used and its performance is comparable to the best (97.0% *vs.* 97.4%). From this perspective, LogME is reasonably useful. In general, how to deal with structured tasks better is a research problem requiring further effort.

Table 4: Ranking pre-trained models in named entity recognition (CoNLL-2003 task).

PTM	RoBERTa	RoBERTa-D	uncased BERT-D	cased BERT-D	ALBERT-v1	ALBERT-v2	ELECTRA-base	ELECTRA-small	τ_w
F-1 score (%)	97.4	96.6	96.8	95.5	97.0	97.4	97.2	91.9	
LogME	0.685	0.723	0.783	0.623	0.834	0.809	0.746	0.646	0.20

6.3 Tuning pre-trained models

This section turns to the second part of the proposed paradigm: tuning pre-trained models. As mentioned in Section 5, most academic researchers are not constrained by the inference

cost of deployed models, and they can use the best-ranked (according to the LogME value) PTM straightforwardly. This paper is concerned with the practical usage scenario, where computational constraints require us to use a specific PTM but we still want to leverage the knowledge from other PTMs in the pre-trained model hub.

The experiments in this section are designed to compare three methods of tuning multiple PTMs: the knowledge distillation approach, the Zoo-tuning approach and the proposed B-Tuning method. We first conduct experiments with multiple homogeneous PTMs where all three methods are applicable, then we dive into the practical case of multiple heterogeneous PTMs. By default, the temperature scaling hyper-parameter t is set to 0.1 in Equation 5.

6.3.1 Tuning multiple homogeneous PTMs

We use five homogeneous pre-trained models following the experimental setup of Zoo-tuning (Shu et al., 2021). They are ResNet-50 models trained by different pre-training tasks: (1) Supervised pre-trained on ImageNet (He et al., 2016); (2) Unsupervised pre-trained by MoCo (He et al., 2020); (3) MaskRCNN model (He et al., 2017); (4) DeepLab V3 (Chen et al., 2017); (5) KeyPoint detection model pre-trained on COCO (Lin et al., 2014). The dataset we use is Aircraft (Maji et al., 2013), the first dataset (alphabetically) in Section 6.2.1. The target model is ResNet-50 pre-trained in ImageNet, following the setting of Shu et al. (2021).

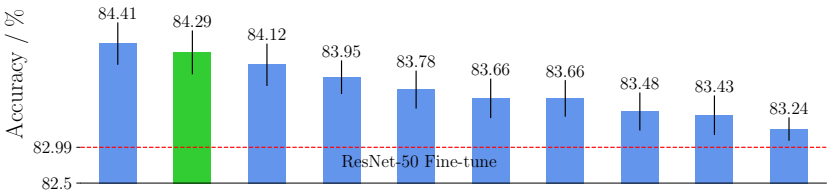
To demonstrate the effectiveness of B-Tuning, we use all five PTMs as the teacher models, and report the performance of three methods (B-Tuning, knowledge distillation, and Zoo-tuning) on multiple PTM tuning in the first row of Table 5. Zoo-tuning performs better than vanilla knowledge distillation, but the new B-Tuning method surpasses Zoo-tuning, setting a new state-of-the-art benchmark for multiple PTM tuning.

Table 5: Accuracy (%) of multiple PTM tuning in Aircraft, with different teacher models and tuning methods. As a baseline, single PTM fine-tuning yields 82.99% accuracy.

method / teacher models	Knowledge Distillation	Zoo-tuning	B-Tuning
all PTMs from the PTM hub	82.97 ± 0.27	83.32 ± 0.32	83.49 ± 0.17
top-3 PTMs (ranked by LogME)	84.29 ± 0.30	-	85.12 ± 0.15

To demonstrate the effectiveness of LogME selection in multiple PTM tuning, we rank five PTMs by LogME, and select the top-K PTMs as the teacher models in the subsequent tuning. Shu et al. (2021) used all five PTMs to tune the target PTM, since they do not investigate how to select PTMs. To sufficiently test the effect of selection, we choose $K = \arg\max_{3 \le K \le 5} \binom{5}{K} = 3$, so that there are many possible selections and later we can explore how optimal LogME selection is. The results are in the second row of Table 5. Surprisingly, selecting top-3 PTMs brings a significant performance improvement, demonstrating the effectiveness of the *"ranking and tuning pre-trained models"* paradigm.

To evaluate the optimality of LogME selection, we try all the $\binom{5}{3} = 10$ combinations of selecting three PTMs from five PTMs. Vanilla knowledge distillation is used to avoid confounders. Results are shown in Figure 10, with the accuracy of fine-tuning a single ResNet-50 as the baseline. We have two observations from Figure 10: (1) Transferring the knowledge from multiple PTMs consistently outperforms fine-tuning a single pre-trained



ImageNet Sup.	0.947	✓	✓	✓		✓	✓				✓
MaskRCNN PT.	0.936	✓	✓		✓		✓		✓	✓	
MoCo PT.	0.934		✓	✓	✓	✓		✓	✓		
KeyPoint PT.	0.914					✓	✓	✓	✓	✓	✓
DeepLab PT.	0.913	✓		✓	✓			✓		✓	✓
PTM name	LogME										

Figure 10: Accuracy of knowledge distillation with three PTM teachers. All $\binom{5}{3} = 10$ combinations of selecting 3 PTM teachers are reported. Selecting top-3 PTMs according to LogME achieves the second best performance among ten combinations.

model (82.99%), which adheres to our intuition that utilizing the rich knowledge from various PTMs is better than fine-tuning alone. (2) The best combination achieves 84.41% accuracy, but usually it is too expensive to try all the combinations (10 trials). Instead, we can use LogME to select the top-3 PTMs, which achieves 84.29% accuracy and is the second best. Moreover, we can select the top-3 PTMs by LogME and then perform B-Tuning, which even surpasses the best combination and has an accuracy of 85.12%.

We can draw three conclusions from experiments in this section: (1) multiple PTM tuning is better than single PTM fine-tuning; (2) it is better (near-optimal among all the possible selections) to select top-ranked PTMs according to LogME than to use all the PTMs; (3) B-Tuning is superior to knowledge distillation and Zoo-tuning.

It is important to point out that selection based on LogME value is a greedy procedure, and this procedure could fail to capture complicated high-order interactions among PTMs. For example, in Figure 10, DeepLab pre-trained model has the lowest LogME value, but it appears in the best combination. How to analyze the high-order interactions among PTMs would be a worthwhile research question in the future.

6.3.2 Tuning multiple heterogeneous PTMs

Section 6.3.1 studies multiple PTM tuning with homogeneous models, which follows the setting of Shu et al. (2021) and demonstrates the superiority of B-Tuning. Nonetheless, compared with tuning multiple homogeneous PTMs, a more general and more attractive application of multiple PTM tuning is to transfer knowledge from a large hub of heterogeneous PTMs. This section focuses on the latter setting, and provides some guidelines on how to select a proper number of PTMs (*i.e.*, the hyper-parameter K) as teachers.

The alphabetically first and second datasets (Aircraft and Birdsnap) are chosen and the PTM hub consists of the 12 PTMs used in Section 6.2.1. The 12 PTMs are ranked by their

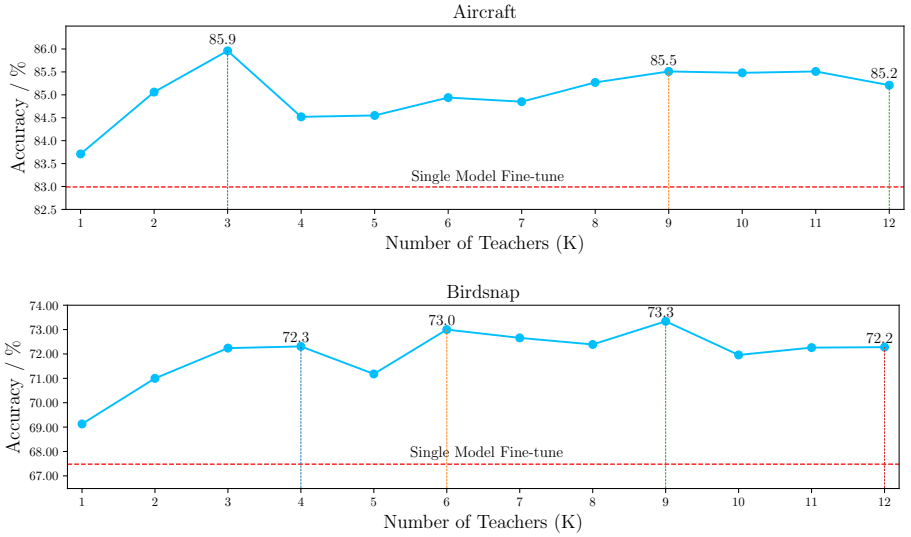


Figure 11: A study on the number of PTMs (K) to use in B-Tuning with two datasets (Top: Aircraft; Bottom: Birdsnap).

LogME values, and the target PTM ϕ_t is the most common ResNet-50. Top-K PTMs are used in B-Tuning to fine-tune the target model, with K varying from 1 to 12. Results are plotted in Figure 11, where the X-axis is the value of K.

We make the following observations based on Figure 11: (1) B-Tuning with multiple PTMs is consistently better than single PTM fine-tuning. (2) B-Tuning with all the 12 PTMs does not yield the best accuracy, which emphasizes the importance of selecting proper PTMs. (3) The trend of accuracy with respect to K is complicated, and how to select an optimal K is a worthwhile topic for future research.

For practitioners, there are two concerns about the choice of K: (1) choosing the optimal K can yield the best accuracy; (2) but larger K incurs a much larger computational cost, since a forward pass of each PTM during tuning is required. Considering the results in Figure 11 and the trade-off between the computational cost and the performance improvement, we recommend choosing K from $\{2, 3, 4\}$ in practice.

6.4 Using ImageNet-1K as the downstream task

The above experiments focus on small-scale and medium-scale downstream tasks, which are common in transfer learning research. This section takes a step further to use the large-scale ImageNet-1K (Deng et al., 2009) as the downstream dataset. In this case, a dataset larger than ImageNet-1K should be used for pre-training. JFT-300M (Sun et al., 2017), Instagram-1B (Mahajan et al., 2018), and ImageNet-21K (Deng et al., 2009) are commonly-used datasets that are larger than ImageNet-1K. Among them, ImageNet-21K is the only publicly available dataset, which serves as the pre-training dataset here. ImageNet-21K pre-trained models are provided by the timm project. It mainly contains models pre-trained on ImageNet-1K, but also has three models pre-trained on ImageNet-21K and fine-tuned

on ImageNet-1K, including MLP-Mixer (Tolstikhin et al., 2021), ViT (Dosovitskiy et al., 2021), and Swin-T (Liu et al., 2021b). With ImageNet-1K as the downstream dataset, their LogME score and fine-tuned reference transfer learning performance are presented in Table 6. LogME is perfectly aligned with the reference performance, with a correlation value $\tau_w = 1$. Then we use B-Tuning to tune the commonly used ResNet-50 trained in ImageNet-1K, with ViT and Swin-T as teacher models. The accuracy is increased from 76.15% to 76.50%.

Experiments in this section demonstrate that LogME and B-Tuning work for not only small-scale and medium-scale datasets but also for large-scale datasets.

Table 6: Ranking models pre-trained on ImageNet-21K transferred to ImageNet-1K.

PTM pre-trained on ImageNet-21K	MLP-Mixer	ViT	Swin-T	τ_w
Fine-tuned Accuracy on ImageNet-1K (%)	76.61	84.53	85.25	1.00
LogME value	2.075	2.085	2.134	

6.5 Efficiency of LogME

A theoretically sound algorithm is often complex and computationally expensive, and this is the case for LogME without optimization. Fortunately, we successfully reduced the computational complexity after analyzing the theoretical convergence of LogME by the fixed point iteration (see Section 4.1). We present a summary of the algorithmic complexity in Table 2, and present empirical results in Table 7, where we show the wall-clock-time speedup measured in Aircraft with ResNet-50. The naïve implementation is very slow. Our conference paper (You et al., 2021) proposed an optimization scheme for matrix multiplication and matrix inversion, which brings 61.7× speedup. This paper further proposes the fixed point iteration algorithm, which results in a much larger speedup (131.5×). Thanks to the optimized method, LogME is not only theoretically sound but also computationally efficient.

Table 7: Quantitative measurement of computational speedup in evidence maximization.

	Wall-clock time (second)	Speedup
evidence maximization (naïve implementation)	802.5 ± 5.6	-
evidence maximization (optimized by You et al. (2021))	13.1 ± 0.7	61.7×
evidence maximization (fixed point iteration, proposed)	6.1 ± 0.7	131.5×

Next, we quantitatively measure the wall-clock and memory footprint of LogME in both computer vision and natural language processing; see Table 8. ResNet 50 on Aircraft is used for computer vision, and RoBERTa-D on MNLI task is used for NLP. The cost for the rest of the models and datasets varies, but the proportion is similar. The cost of computing reference transferability T_m (fine-tuning with hyper-parameter search) serves as the upper bound of ranking pre-trained models. Note that, because carelessly tuned hyper-parameters cannot tell good models apart from bad models, it is necessary to attribute the cost of hyper-parameter search to fine-tuning. We also list the cost of extracting features by pre-trained models, which is the lower bound of ranking pre-trained models.

Table 8: Computational cost and memory footprint of LogME.

	wall-clock time		memory footprint	
Computer Vision	fine-tune (upper bound)	161000s	fine-tune (upper bound)	6.3 GB
	extract feature (lower bound)	37s	extract feature (lower bound)	43 MB
	LogME	43s	LogME	53 MB
	benefit	3700 ↑	benefit	120 ↑
Natural Language Processing	fine-tune (upper bound)	100200s	fine-tune (upper bound)	88 GB
	extract feature (lower bound)	1130s	extract feature (lower bound)	1.2 GB
	LogME	1136s	LogME	1.2 GB
	benefit	88 ↑	benefit	73 ↑

Based on Table 8, we have the following observations: (1) brute-force fine-tuning is computationally expensive, requiring about a day for one dataset with one pre-trained model. Selecting the best pre-trained model out of 12 models would cost 12 GPU-days. (2) Extracting features is very cheap and costs much less than fine-tuning. (3) The additional time-cost of LogME compared to feature extraction is rather small, which means that *LogME's cost is very close to the lower bound*. In computer vision, LogME is 3700× faster than fine-tuning, with 120× less memory footprint. In the NLP domain, feature extraction is much slower than that in computer vision, and therefore the wall-clock time speedup (88×) is not that striking.

In summary, LogME is efficient in terms of both wall-clock time and memory footprint, thanks to the optimized algorithm (fixed point iteration) inspired by the theoretical analysis.

6.6 Comparing LogME to re-training head

A straightforward way to measure the relationship between features and labels is to train a linear classification/regression head for the downstream task, and to use the head's performance as a metric, which is known as "*linear probing*" or "*linear protocol evaluation*." Empirically, we find that re-training the head does not work well. In the following, we summarize why re-training the head is inferior to LogME from three perspectives, which partially explains why the important problem of ranking and tuning PTMs was under-explored in the past.

(1) LogME is more efficient than re-training head. In linear protocol evaluation, parameters in the head are learned by maximum likelihood estimation, which is prone to over-fitting. To alleviate over-fitting, grid search for its hyper-parameters (such as the strength of L2 regularization) should be tuned extensively on a validation set, making head re-training inefficient. For example, in the Caltech dataset, we extract features from 12 PTMs, train softmax regressors with tuned hyper-parameters (the L2 regularization strength), and plot the correlation τ_w between the best head accuracy and the reference transfer performance with respect to the number of hyper-parameter trials in Figure 12. The correlation of LogME is plotted as a reference. Computing LogME requires 3× less time than re-training a head with one fixed hyper-parameter, and re-training a head with exhaustive hyper-parameter search is still much inferior to LogME.

(2) Re-training head does not work well with limited training data. Because re-training the head follows a supervised learning paradigm, it suffers in low-shot learning

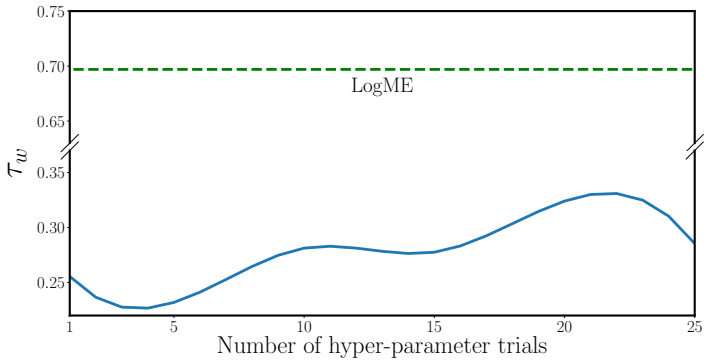


Figure 12: The correlation of re-training the head with respect to the number of hyper-parameter trials.

scenarios. For example, prompt learning (Liu et al., 2021a) is an active research area in natural language processing, where researchers try to exploit the potential of frozen pre-trained models with only a few training data points. In the sentiment classification task SST-2 (Socher et al., 2013), prompt learning (Liu et al., 2021a) extracts the sentence embedding E_S for each sentence S, and compares E_S with word embeddings E_P, E_N of a positive anchor word P (such as "good" and "fantastic") and a negative anchor word N (such as "bad" and "awful"). The decision rule is: `sentence S contains positive sentiment` $\iff E_S^T E_P > E_S^T E_N$. In this case, searching for proper anchor words on validation data yields 61.4% accuracy (complete results are available in Table 13). Re-training the head (*i.e.*, training a simple classification head with limited training data on the frozen sentence embedding E_S and tuning the weight-decay hyper-parameter on a validation set) only achieves 51.64% accuracy with 10 sentences for training. Meanwhile, we can apply LogME (or to be specific, the posterior predictive distribution introduced in Section 5.3) to this problem, which does not require any hyper-parameter tuning. This way, we can combine training data with validation data to compute the predictive weight m for each class, and use m as the embedding of a virtual "anchor word," which results in 79.24% accuracy, *a huge improvement over re-training head and manually selected anchor words*. Moreover, the superior performance of LogME is interpretable: we analyzed the predictive weight m for the negative sentiment class, and find that it is closest to the embedding of "`dump`," "`:(`," "`doomed`," "`Worse`," and "`worse`." Interestingly, it can discover that "`:(`," a cyber word used to express unhappy emotion, contains negative sentiment.

(3) Re-training the head does not have a clear metric. As a side issue, even if we re-train a head for a downstream task, it is unclear which quantity should be used as the ranking metric. When the performance of a downstream task is evaluated by accuracy or MSE, is it over-fitting to use the accuracy or MSE of the re-trained head? Indeed, in Figure 12, when the number of hyper-parameter trials increases, the correlation can even go down, confirming the concern of over-fitting. In contrast, LogME is based on unified modeling of label evidence, which has clear statistical support.

7. Conclusions

Pre-trained models are universally acknowledged as a foundation of deep learning. Researchers have explored many ways to create and exploit PTMs. In this paper, we change the focus from individual PTMs to PTM hubs, and study how to sufficiently exploit PTM hubs within a new paradigm of ranking and tuning pre-trained models. The ranking part introduces a theoretically sound and computationally efficient transferability metric named LogME. LogME is then further extended to be a multiple PTM tuning method that we refer to as B-Tuning, which completes the tuning part of the paradigm.

We presented extensive experiments that confirm the effectiveness of the proposed methods in ranking (LogME *vs.* brute-force fine-tuning/LEEP/NCE), selection (top-K PTMs by LogME *vs.* exponentially many combinations), and tuning (B-Tuning *vs.* Zoo-tuning and Knowledge Distillation), showing that the new paradigm of exploiting PTM hubs is attractive for practitioners.

Acknowledgments

We would like to present our thanks to Ximei Wang, Xinyang Chen, Yang Shu at Tsinghua University, Yi Zeng at Peking University, and Yonglong Tian at MIT for helpful discussions. Kaichao You, Yong Liu, Jianmin Wang, and Mingsheng Long are supported by the National Key Research and Development Project (2021YFB1715200), the National Natural Science Foundation of China (62022050 and 62021002), the Beijing Nova Program (Z201100006820041), the BNRist Scholar Fund (BNR2021RC01002), and the Tsinghua-Huawei Innovation Fund.

References

S. Ben-David and R. Schuller. Exploiting task relatedness for multiple task learning. In *COLT*, pages 567–580, Berlin, Heidelberg, 2003.

T. Berg, J. Liu, S. Woo Lee, M. L. Alexander, D. W. Jacobs, and P. N. Belhumeur. Birdsnap: Large-scale fine-grained visual categorization of birds. In *CVPR*, pages 2011–2018, Columbus, Ohio, 2014.

C. M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, New York, 2006.

R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, E. Brynjolfsson, S. Buch, D. Card, R. Castellon, N. Chatterji, A. Chen, K. Creel, J. Q. Davis, D. Demszky, C. Donahue, M. Doumbouya, E. Durmus, S. Ermon, J. Etchemendy, K. Ethayarajh, L. Fei-Fei, C. Finn, T. Gale, L. Gillespie, K. Goel, N. Goodman, S. Grossman, N. Guha, T. Hashimoto, P. Henderson, J. Hewitt, D. E. Ho, J. Hong, K. Hsu, J. Huang, T. Icard, S. Jain, D. Jurafsky, P. Kalluri, S. Karamcheti, G. Keeling, F. Khani, O. Khattab, P. W. Kohd, M. Krass, R. Krishna, R. Kuditipudi, A. Kumar, F. Ladhak, M. Lee, T. Lee, J. Leskovec, I. Levent, X. L. Li,

X. Li, T. Ma, A. Malik, C. D. Manning, S. Mirchandani, E. Mitchell, Z. Munyikwa, S. Nair, A. Narayan, D. Narayanan, B. Newman, A. Nie, J. C. Niebles, H. Nilforoshan, J. Nyarko, G. Ogut, L. Orr, I. Papadimitriou, J. S. Park, C. Piech, E. Portelance, C. Potts, A. Raghunathan, R. Reich, H. Ren, F. Rong, Y. Roohani, C. Ruiz, J. Ryan, C. Ré, D. Sadigh, S. Sagawa, K. Santhanam, A. Shih, K. Srinivasan, A. Tamkin, R. Taori, A. W. Thomas, F. Tramèr, R. E. Wang, W. Wang, B. Wu, J. Wu, Y. Wu, S. M. Xie, M. Yasunaga, J. You, M. Zaharia, M. Zhang, T. Zhang, X. Zhang, Y. Zhang, L. Zheng, K. Zhou, and P. Liang. On the Opportunities and Risks of Foundation Models. *arXiv:2108.07258 [cs]*, 2021.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners. In *NeurIPS*, pages 1877–1901, 2020.

Z. Cao, K. You, Z. Zhang, J. Wang, and M. Long. From Big to Small: Adaptive Learning to Partial-Set Domains. *TPAMI*, 2022. early access.

L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv:1706.05587 [cs]*, 2017.

T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, pages 1597–1607, 2020a.

X. Chen, S. Wang, B. Fu, M. Long, and J. Wang. Catastrophic Forgetting Meets Negative Transfer: Batch Spectral Shrinkage for Safe Transfer Learning. In *NeurIPS*, page 1906–1916, Vancouver, Canada, 2019.

X. Chen, H. Fan, R. Girshick, and K. He. Improved baselines with momentum contrastive learning. *arXiv:2003.04297 [cs]*, 2020b.

M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *CVPR*, page 3606–3613, Columbus, Ohio, 2014.

K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning. ELECTRA: Pre-training text encoders as discriminators rather than generator. In *ICLR*, 2020.

T. M. Cover. *Elements of Information Theory*. John Wiley & Sons, 1999.

J. Daunizeau. Semi-analytical approximations to statistical moments of sigmoid and softmax mappings of normal variables. *arXiv preprint arXiv:1703.00091*, 2017.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1):1–22, 1977.

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, page 248–255, Miami Beach, Florida, 2009.

J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, page 4171–4186, Minneapolis, Minnesota, 2019.

J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *ICML*, page 647–655, Beijing, China, 2014.

A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

D. Erhan, A. Courville, Y. Bengio, and P. Vincent. Why does unsupervised pre-training help deep learning? In *AISTATS*, page 201–208, Sardinia, Italy, 2010.

R. Fagin, R. Kumar, and D. Sivakumar. Comparing top k lists. In *SODA*, page 28–36, Baltimore, Maryland, 2003.

L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *CVPR Workshops*, page 178–178, Washington D.C., 2004.

Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. In *ICML*, pages 1180–1189, Lille, France, 2015.

R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, page 580–587, Columbus, Ohio, 2014.

S. F. Gull. Developments in maximum entropy data analysis. In *Maximum Entropy and Bayesian Methods*. 1989.

M. Gutmann and A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *AISTATS*, pages 297–304, Sardinia, Italy, 2010.

X. Han, Z. Zhang, N. Ding, Y. Gu, X. Liu, Y. Huo, J. Qiu, L. Zhang, W. Han, M. Huang, Q. Jin, Y. Lan, Y. Liu, Z. Liu, Z. Lu, X. Qiu, R. Song, J. Tang, J.-R. Wen, J. Yuan, W. X. Zhao, and J. Zhu. Pre-trained models: past, present and future. *arXiv:2106.07139 [cs]*, 2021.

G. H. Hardy, J. E. Littlewood, G. Pólya, and G. Pólya. *Inequalities*, volume 30. Springer Science & Business Media, 1952.

K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, page 1026–1034, Santiago, Chile, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, page 770–778, Las Vegas, Nevada, 2016.

K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. In *ICCV*, pages 2980–2988, Venice, Italy, 2017.

K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, pages 9729–9738, 2020.

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

W. Hu, B. Liu, J. Gomes, M. Zitnik, P. Liang, V. Pande, and J. Leskovec. Strategies for pre-training graph neural networks. In *ICLR*, 2020.

G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *CVPR*, pages 4700–4708, Hawaii, USA, 2017.

A. Immer, M. Bauer, V. Fortuin, G. Rätsch, and K. M. Emtiyaz. Scalable marginal likelihood estimation for model selection in deep learning. In *ICML*, page 4563–4573, 2021.

L. Jing and Y. Tian. Self-supervised visual feature learning with deep neural networks: A survey. *TPAMI*, 43(11):4037–4058, 2020.

N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, and A. Borchers. In-datacenter performance analysis of a tensor processing unit. In *ISCA*, pages 1–12, Toronto, Canada, 2017.

M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1):81–93, 1938.

K. H. Knuth, M. Habeck, N. K. Malakar, A. M. Mubeen, and B. Placek. Bayesian evidence and model selection. *Digital Signal Processing*, 47:50–67, 2015.

D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, Cambridge, Massachusetts, 2009.

S. Kornblith, J. Shlens, and Q. V. Le. Do better imagenet models transfer better? In *CVPR*, page 2661–2671, Los Angeles, California, 2019.

Z. Kou, K. You, M. Long, and J. Wang. Stochastic normalization. In *NeurIPS*, pages 16304–16314, 2020.

J. Krause, J. Deng, M. Stark, and L. Fei-Fei. Collecting a large-scale dataset of fine-grained cars. Technical report, 2013.

A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, 2009.

Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. ALBERT: A lite BERT for self-supervised learning of language representations. In *ICLR*, 2020.

C. Li, Y. Mao, R. Zhang, and J. Huai. On hyper-parameter estimation in empirical Bayes: a revisit of the MacKay algorithm. In *UAI*, page 477–486, Arlington, Virginia, 2016.

H. Li, P. Chaudhari, H. Yang, M. Lam, A. Ravichandran, R. Bhotika, and S. Soatto. Rethinking the hyperparameters for fine-tuning. In *ICLR*, 2020.

X. Li, Y. Grandvalet, and F. Davoine. Explicit inductive bias for transfer learning with convolutional networks. In *ICML*, pages 2825–2834, Stockholm, Sweden, 2018.

T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, page 740–755, Zurich, Switzerland, 2014.

P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig. Pre-train, Prompt, and Predict: A systematic survey of prompting methods in natural language processing. *arXiv:2107.13586 [cs]*, 2021a.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows. In *ICCV*, pages 10012–10022, Montreal, Canada, 2021b.

M. Long, Y. Cao, J. Wang, and M. Jordan. Learning transferable features with deep adaptation networks. In *ICML*, pages 97–105, Lille, France, 2015.

D. J. MacKay. Bayesian interpolation. *Neural Computation*, 4(3):415–447, 1992.

D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten. Exploring the limits of weakly supervised pretraining. In *ECCV*, page 181–196, Munich, Germany, 2018.

S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. *arXiv:1306.5151 [cs]*, 2013.

L. Matthey, I. Higgins, D. Hassabis, and A. Lerchner. dSprites: Disentanglement testing Sprites dataset. Technical report, 2017.

S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. In *ICLR*, Toulon, France, 2017.

B. Neyshabur, H. Sedghi, and C. Zhang. What is being transferred in transfer learning? In *NeurIPS*, pages 512–523, 2020.

C. Nguyen, T. Hassner, M. Seeger, and C. Archambeau. LEEP: A new measure to evaluate transferability of learned representations. In *ICML*, pages 7294–7305, 2020.

O. M. Parkhi, A. Vedaldi, A. Zisserman, and C. V. Jawahar. Cats and dogs. In *CVPR*, pages 3498–3505, Providence, Rhode Island, 2012.

X. Qiu, T. Sun, Y. Xu, Y. Shao, N. Dai, and X. Huang. Pre-trained models for natural language processing: A survey. *Science China Technological Sciences*, 63(10):1872–1897, 2020.

J. Quionero-Candela, M. Sugiyama, A. Schwaighofer, and N. D. Lawrence. *Dataset Shift in Machine Learning*. MIT Press, Cambridge, Massachusetts, 2009.

P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. SQuAD: 100,000+ Questions for Machine Comprehension of Text. In *EMNLP*, page 2383–2392, Austin, Texas, 2016.

C. E. Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71, Berlin, Heidelberg, 2003.

O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, and M. Bernstein. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. MobileNetV2: inverted residuals and linear bottlenecks. In *CVPR*, pages 4510–4520, Salt Lake City, Utah, 2018.

E. T. K. Sang and F. De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *NAACL*, page 142–147, Edmonton, Canada, 2003.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv:1910.01108*, 2019.

Y. Shu, Z. Kou, Z. Cao, J. Wang, and M. Long. Zoo-Tuning: Adaptive transfer from a zoo of models. In *ICML*, pages 9626–9637, 2021.

S. P. Singh and M. Jaggi. Model fusion via optimal transport. In *NeurIPS*, pages 22045–22055, 2020.

R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*, page 1631–1642, Seattle, USA, 2013.

C. Sun, A. Shrivastava, S. Singh, and A. Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *ICCV*, page 843–852, Venice, Italy, 2017.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. pages 1–9, Boston, Massachusetts, 2015.

C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. In *CVPR*, pages 2818–2826, Las Vegas, Nevada, 2016.

M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, page 2820–2828, Los Angeles, California, 2019.

S. Thrun and L. Pratt. Learning to Learn: Introduction and Overview. In *Learning to Learn*, pages 3–17, Boston, Massachusetts, 1998.

Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola. What makes for good views for contrastive learning? In *NeurIPS*, pages 6827–6839, 2020.

I. O. Tolstikhin, N. Houlsby, A. Kolesnikov, L. Beyer, X. Zhai, T. Unterthiner, J. Yung, A. Steiner, D. Keysers, and J. Uszkoreit. Mlp-mixer: An all-mlp architecture for vision. In *NeurIPS*, pages 24261–24272, 2021.

A. T. Tran, C. V. Nguyen, and T. Hassner. Transferability and hardness of supervised classification tasks. In *ICCV*, page 1395–1405, Seoul, Korea, 2019.

S. Vigna. A weighted correlation index for rankings with ties. In *WWW*, pages 1166–1176, Florence, Italy, 2015.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *EMNLP*, page 353–355, Brussels, Belgium, 2018.

A. Wang, Y. Pruksachatkun, N. Nangia, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman. SuperGLUE: A stickier benchmark for general-purpose language understanding systems. In *NeurIPS*, Vancouver, Canada, 2019.

T. Wolf, J. Chaumond, L. Debut, V. Sanh, C. Delangue, A. Moi, P. Cistac, M. Funtowicz, J. Davison, and S. Shleifer. Transformers: State-of-the-art natural language processing. In *EMNLP*, pages 38–45, 2020.

J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *CVPR*, pages 3485–3492, San Francisco, California, 2010.

Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *NeurIPS*, Vancouver, Canada, 2019.

J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *NeurIPS*, page 3320–3328, Montreal, Canada, 2014.

K. You, Z. Kou, M. Long, and J. Wang. Co-tuning for transfer learning. In *NeurIPS*, pages 17236–17246, 2020.

K. You, Y. Liu, J. Wang, and M. Long. LogME: Practical assessment of pre-trained models for transfer learning. In *ICML*, pages 12133–12143, 2021.

S. Zagoruyko and N. Komodakis. Wide residual networks. In *BMVC*, York, UK, 2016.

A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling Task Transfer Learning. In *CVPR*, pages 3712–3722, Salt Lake City, Utah, 2018.

X. Zhai, J. Puigcerver, A. Kolesnikov, P. Ruyssen, C. Riquelme, M. Lucic, J. Djolonga, A. S. Pinto, M. Neumann, A. Dosovitskiy, L. Beyer, O. Bachem, M. Tschannen, M. Michalski, O. Bousquet, S. Gelly, and N. Houlsby. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv:1910.04867 [cs, stat]*, 2020.

Appendix A. Notation Table

Notation used in this paper is listed in the following table. We endeavored to avoid notational conflicts, but the following conflicts are worth noting: (1) f_i represents features of x_i and $f(t)$ represents the fixed point iteration function. (2) w represents parameters in the linear head while the weighted Kendall's rank correlation τ_w uses w as its subscript. (3) t is used in the convergence proof and also used for the temperature hyper-parameter in B-Tuning.

Table 9: Notations used in this paper.

notation	dimensionality	meaning
i, j, k	$\mathbb{N}$	running subscripts
M, n	$\mathbb{N}$	the number of PTMs and samples
K	$\mathbb{N}$	the number of selected PTMs for subsequent tuning
C, D	$\mathbb{N}$	the dimension of label and extracted feature
ϕ	–	a pre-trained model
x_i	–	an input sample
$f_i = \phi(x_i)$	$\mathbb{R}^D$	extracted feature of an input example
$F = [f_1, \ldots, f_n]^T$	$\mathbb{R}^{n \times D}$	stacked features of f_i
Y_i	$\mathbb{R}^C$	label of x_i
y_i	$\mathbb{R}$	a component of Y_i
y	$\mathbb{R}^n$	the label component for all n samples
y'	–	predictive distribution of an input sample
$\pi_k = \frac{\exp(\mathcal{L}_k/t)}{\sum_{j=1}^K \exp(\mathcal{L}_j/t)}$	$\mathbb{R}$	weighted coefficient for each teacher model ϕ_k
$\bar{y}' = \sum_{k=1}^K \pi_k y'_k$	–	weighted average of y'
T	$\mathbb{R}$	reference transfer performance
S	$\mathbb{R}$	score produced by a transferability metric
τ	$\mathbb{R}$	Kendall's rank correlation
τ_w	$\mathbb{R}$	weighted Kendall's rank correlation
w	$\mathbb{R}^D$	parameter in linear head
α, β	$\mathbb{R}$	hyper-parameter of the Bayesian linear model
$A = \alpha I_D + \beta F^T F$	$\mathbb{R}^{D \times D}$	a quantity in calculating LogME
$m = \beta A^{-1} F^T y$	$\mathbb{R}^D$	a quantity in calculating LogME
γ	$\mathbb{R}$	a quantity in calculating LogME
$\mathcal{L} = \mathcal{L}(\alpha, \beta)$	$\mathbb{R}$	log evidence given α, β
α^*, β^*	$\mathbb{R}$	α, β to achieve maximum evidence
U, Σ, V	–	matrices in SVD ($F = U\Sigma V^T$)
σ	$\mathbb{R}$	diagonal entries in Σ
r	$\mathbb{N}$	the rank of matrix F
$z = U^T y$	$\mathbb{R}^n$	transformation of y under U
$t = \frac{\alpha}{\beta}$	$\mathbb{R}$	a quantity in convergence proof
t', α', β'	$\mathbb{R}$	the value of t, α, β after an iteration
$f(t)$	–	the fixed point iteration function
$\tilde{\mathcal{L}}, \tilde{F}$	–	$\mathcal{L}, F$ for duplicated or padded features
W	–	transformation matrix in knowledge distillation

Appendix B. Proof of Theorem 1

Theorem 1: Algorithm 4 induces a scalar function $t' = f(t) = \left(\frac{n}{n-\sum_{i=1}^{D}\frac{\sigma_i^2}{t+\sigma_i^2}} - 1\right) t^2 \frac{\sum_{i=1}^{n}\frac{z_i^2}{(t+\sigma_i^2)^2}}{\sum_{i=1}^{n}\frac{\sigma_i^2 z_i^2}{(t+\sigma_i^2)^2}}$.

Proof Let's express all symbols in a unified form with respect to $\alpha, \beta, \Sigma, z, U, V$:

- $A = \alpha I + \beta F^T F = V(\alpha I + \beta \Sigma^T \Sigma) V^T$
- $A^{-1} = V \Sigma_{inv} V^T$ where $(\Sigma_{inv})_{ii} = \frac{1}{\alpha + \beta\sigma_i^2}$ $(1 \le i \le D)$
- $m = \beta A^{-1} F^T y = \beta V \Sigma_{inv} \Sigma^T z$
- $m^T m = z^T \Sigma_m z$ with $\Sigma_m = \beta^2 \Sigma \Sigma_{inv}^2 \Sigma^T$ and $(\Sigma_m)_{ii} = \frac{\beta^2 \sigma_i^2}{(\alpha+\beta\sigma_i^2)^2}$, so $m^T m = \sum_{i=1}^{n} \frac{\beta^2 \sigma_i^2 z_i^2}{(\alpha+\beta\sigma_i^2)^2}$
- $Fm = \beta U \Sigma \Sigma_{inv} \Sigma^T z$, $Fm - y = U \Sigma_{res} z$ with $\Sigma_{res} = \beta \Sigma \Sigma_{inv} \Sigma^T - I$, $(\Sigma_{res})_{ii} = -\frac{\alpha}{\alpha+\beta\sigma_i^2}$
- $||Fm - y||_2^2 = (Fm-y)^T (Fm-y) = z^T (\Sigma_{res})^2 z = \sum_{i=1}^{n} \frac{\alpha^2 z_i^2}{(\alpha+\beta\sigma_i^2)^2}$
- $\gamma = \sum_{i=1}^{D} \frac{\beta\sigma_i^2}{\alpha+\beta\sigma_i^2} = \sum_{i=1}^{D} \frac{\sigma_i^2}{t+\sigma_i^2}$

Putting them together, we have

$$t' = \frac{\alpha'}{\beta'} = \frac{\gamma}{n-\gamma} \frac{||Fm-y||_2^2}{m^T m} = \left(\frac{n}{n - \sum_{i=1}^{D}\frac{\sigma_i^2}{t+\sigma_i^2}} - 1\right) t^2 \frac{\sum_{i=1}^{n}\frac{z_i^2}{(t+\sigma_i^2)^2}}{\sum_{i=1}^{n}\frac{\sigma_i^2 z_i^2}{(t+\sigma_i^2)^2}} = f(t)$$

■

Appendix C. Proof of Theorem 2

Theorem 2: If $r < n$ and $\sum_{1 \le i,j \le n} (z_i^2 - z_j^2)(\sigma_i^2 - \sigma_j^2) > 0$, then $f(t)$ has a fixed point and thus MacKay's algorithm will converge.

Proof The theorem can be proved by studying the behavior of $f(t)$ near 0 and ∞.

We have $\lim_{t\to 0} f(t) = \frac{r}{n-r} \frac{\sum_{i=r+1}^{n} z_i^2}{\sum_{i=1}^{r} z_i^2} > 0$, which is a constant and positive number.

When t approaches infinity, we find that $\lim_{t\to\infty} \frac{f(t)}{t} = \frac{\sum_{i=1}^{n}\sigma_i^2}{n} \frac{\sum_{i=1}^{n} z_i^2}{\sum_{i=1}^{n}\sigma_i^2 z_i^2}$ is constant, which means $f(t)$ behaves linearly when t is large enough.

Exploiting a trick used in proving the Chebyshev's Sum Inequality (Hardy et al., 1952), we obtain $\sum_{1 \le i,j \le n} (z_i^2 - z_j^2)(\sigma_i^2 - \sigma_j^2) = 2n \sum_{i=1}^{n} \sigma_i^2 z_i^2 - 2(\sum_{i=1}^{n}\sigma_i^2)(\sum_{i=1}^{n} z_i^2)$. The condition $\sum_{1 \le i,j \le n} (z_i^2 - z_j^2)(\sigma_i^2 - \sigma_j^2) > 0$ thus translates into $\frac{\sum_{i=1}^{n}\sigma_i^2}{n} \frac{\sum_{i=1}^{n} z_i^2}{\sum_{i=1}^{n}\sigma_i^2 z_i^2} < 1$, which means $f(t)$ increases linearly with a slope smaller than 1 (*i.e.*, $\lim_{t\to\infty} \frac{f(t)}{t} = \frac{\sum_{i=1}^{n}\sigma_i^2}{n} \frac{\sum_{i=1}^{n} z_i^2}{\sum_{i=1}^{n}\sigma_i^2 z_i^2} < 1$).

In summary, when t approaches 0, it is assured that $\lim_{t\to 0} f(t) > t = 0$; when t is large enough, it is assured that $f(t) < t$. **Putting these two conditions together, we conclude the existence of a fixed point $t_0 > 0$ such that $f(t_0) = t_0$.** ■

Appendix D. Proof of Corollary 3

Corollary 3: The LogME value will remain the same if the feature consists of arbitrary replicas of the original feature. Formally speaking, if the LogME value for $F \in \mathbb{R}^{n\times D}$ and $y \in \mathbb{R}^n$ is $\mathcal{L}$, then the LogME value for $\tilde{F} = [F, ..., F] \in R^{n\times qD}$ and $y \in \mathbb{R}^n$ is also $\mathcal{L}$. ($q \in \mathbb{N}$ is a natural number to represent the number of replicas.)

Proof Since LogME is calculated via an iterative algorithm, we prove the corollary by an iterative invariant (a quantitative relation that holds after every while-loop iteration).

Preliminary: SVD of $\tilde{F}$. We have already known the SVD of F is $F = U\Sigma V^T$, and σ_i is the i-th largest eigenvalue of FF^T. Since $\tilde{F}\tilde{F}^T = qFF^T$, duplicated feature $\tilde{F}$ has singular values $\tilde{\sigma}_i^2 = \begin{cases} q\sigma_i^2 & 1 \le i \le D \\ 0 & D+1 \le i \le qD \end{cases}$, and its left orthogonal matrix is the same as F: $\tilde{U} = U$. The right orthogonal matrix of $\tilde{F}$ is somewhat complicated. Let's find an orthogonal matrix $Q_{q\times q}$, whose entries in the first column are $\frac{1}{\sqrt{q}}$. Entries in the other columns do not matter, as long as $Q_{q\times q}$ is a valid orthogonal matrix. For example, we can use $Q_{2\times 2} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$, and $Q_{3\times 3} = \begin{bmatrix} \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{6}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{3}} & \frac{2}{\sqrt{6}} & 0 \\ \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{6}} & \frac{1}{\sqrt{2}} \end{bmatrix}$. Then the right orthogonal matrix of $\tilde{F}$ is $\tilde{V} = Q_{q\times q} \otimes V$, where $\otimes$ is the Kronecker product of two matrices. Using the block matrix form of Kronecker product, we can write down $\tilde{V}$ as $\tilde{V} = \begin{bmatrix} \frac{1}{\sqrt{p}}V & \dots & \dots \\ \vdots & \ddots & \vdots \\ \frac{1}{\sqrt{p}}V & \dots & \dots \end{bmatrix} \in R^{qD\times qD}$, with the first D columns of $\tilde{V}$ corresponding to singular values $\sqrt{q}\sigma_i, 1 \le i \le D$, and the other $(q-1)\times D$ columns of $\tilde{V}$ are orthogonal basis with respect to singular values $\sigma_i = 0$. In summary, if the SVD of F is $F = U\Sigma V^T$, then the SVD of $\tilde{F} = [F, ..., F]$ is $\tilde{F} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$, where $\tilde{U} = U, \tilde{\Sigma} = [\sqrt{q}\Sigma, 0, ..., 0], \tilde{V} = \begin{bmatrix} \frac{1}{\sqrt{q}}V & \dots & \dots \\ \vdots & \ddots & \vdots \\ \frac{1}{\sqrt{q}}V & \dots & \dots \end{bmatrix} = Q_{q\times q} \otimes V$.

Iterative invariant: if we apply Algorithm 2 to both $\tilde{F}$ and F, with a small change that we initialize $\tilde{\alpha} = q, \tilde{\beta} = 1$, then $\tilde{\alpha} = q\alpha, \tilde{\beta} = \beta$ holds before Line 5. Suppose $\tilde{\alpha} = q\alpha, \tilde{\beta} = \beta$ holds before a while-loop, then we obtain:

$$\tilde{\gamma} = \sum_{i=1}^{qD} \frac{\tilde{\beta}\tilde{\sigma}_i^2}{\tilde{\alpha} + \tilde{\beta}\tilde{\sigma}_i^2} = \sum_{i=1}^{D} \frac{q\beta\sigma_i^2}{q\alpha + q\beta\sigma_i^2} = \sum_{i=1}^{D} \frac{\beta\sigma_i^2}{\alpha + \beta\sigma_i^2} = \gamma$$

$$\tilde{\Lambda} = \operatorname{diag}\left\{\tilde{\alpha} + \tilde{\beta}\tilde{\sigma}_i^2\right\}, \tilde{\alpha} + \tilde{\beta}\tilde{\sigma}_i^2 = \begin{cases} q(\alpha + \beta\sigma_i^2) & 1 \le i \le D \\ q\alpha & D+1 \le i \le qD \end{cases}$$

$$
\begin{aligned}
\tilde{m} &= \tilde{\beta}\tilde{A}^{-1}\tilde{F}^T y = \beta\tilde{V}\tilde{\Lambda}^{-1}\tilde{V}^T\tilde{V}\tilde{\Sigma}^T\tilde{U}^T y = \beta\tilde{V}\tilde{\Lambda}^{-1}\tilde{\Sigma}^T U^T y \\
&= \beta\left(\begin{bmatrix} \frac{1}{\sqrt{q}}V & \cdots & \cdots \\ \vdots & \ddots & \vdots \\ \frac{1}{\sqrt{q}}V & \cdots & \cdots \end{bmatrix}\begin{bmatrix} \frac{1}{q}\Lambda^{-1} & \\ & \frac{1}{q\alpha}I_{(q-1)\times D} \end{bmatrix}\begin{bmatrix} \sqrt{q}\Sigma^T \\ 0 \\ \cdots \\ 0 \end{bmatrix}\right)U^T y \\
&= \begin{bmatrix} \frac{1}{q}V\Lambda^{-1}U^T y \\ \cdots \\ \frac{1}{q}V\Lambda^{-1}U^T y \end{bmatrix} = \begin{bmatrix} \frac{1}{q}m \\ \cdots \\ \frac{1}{q}m \end{bmatrix}.
\end{aligned}
$$

Therefore $\tilde{m}^T\tilde{m} = \frac{1}{q}m^T m, \tilde{F}\tilde{m} = [F, \ldots, F]\begin{bmatrix} \frac{1}{q}m \\ \cdots \\ \frac{1}{q}m \end{bmatrix} = Fm.$

After the while-loop iteration, $\tilde{\alpha}' = \frac{\tilde{\gamma}}{\tilde{m}^T\tilde{m}} = \frac{\gamma}{\frac{1}{q}m^T m} = q\alpha', \tilde{\beta}' = \frac{n-\tilde{\gamma}}{||\tilde{F}\tilde{m}-y||_2^2} = \frac{n-\gamma}{||Fm-y||_2^2} = \beta'$, then the iterative invariant $\tilde{\alpha} = q\alpha, \tilde{\beta} = \beta$ still holds. Therefore we know that when the algorithm converges, $\tilde{\alpha}^* = q\alpha^*, \tilde{\beta}^* = \beta^*$. The corresponding maximum evidence is

$$
\begin{aligned}
\tilde{\mathcal{L}} &= \frac{n}{2}\log\tilde{\beta}^* + \frac{qD}{2}\log\tilde{\alpha}^* - \frac{n}{2}\log 2\pi - \frac{\tilde{\beta}^*}{2}||\tilde{F}\tilde{m}-y||_2^2 - \frac{\tilde{\alpha}^*}{2}\tilde{m}^T\tilde{m} - \frac{1}{2}\log\left|\tilde{A}^*\right| \\
&= \frac{n}{2}\log\beta^* + \frac{qD}{2}\log(q\alpha^*) - \frac{n}{2}\log 2\pi - \frac{\beta^*}{2}||Fm-y||_2^2 - \frac{\alpha^*}{2}m^T m - \frac{1}{2}\log\left|\tilde{\Lambda}^*\right| \\
&= \frac{n}{2}\log\beta^* + \frac{qD}{2}\log\left(q\alpha^*\right) - \frac{n}{2}\log 2\pi - \frac{\beta^*}{2}||Fm-y||_2^2 - \frac{\alpha^*}{2}m^T m \\
&\quad - \frac{1}{2}\log|\Lambda^*| - \frac{1}{2}\log\left(q^D\left(q\alpha^*\right)^{(q-1)D}\right) \\
&= \mathcal{L} - \frac{D}{2}\log\alpha^* + \frac{qD}{2}\log(q\alpha^*) - \frac{1}{2}\log\left(q^D\left(q\alpha^*\right)^{(q-1)D}\right) \\
&= \mathcal{L}.
\end{aligned}
$$

By the convergence analysis in Section 4.1, initialization of α, β only changes the initial value of t, which does not impact the convergence value of the fixed point iteration. Therefore, we can conclude that duplicating features will not change the value of LogME.

Although the above proof targets Algorithm 2, it is straightforward to adapt the proof to Algorithm 3. ■

Appendix E. Proof of Corollary 4

Corollary 4: The LogME value will remain the same if the feature is padded with arbitrary number of zeros. Formally speaking, if the LogME value for $F \in \mathbb{R}^{n\times D}$ and $y \in \mathbb{R}^n$ is $\mathcal{L}$, then the LogME value for $\tilde{F} = [F, \mathbf{0}] \in R^{n\times(D+d)}$ and $y \in \mathbb{R}^n$ is also $\mathcal{L}$. $d \in \mathbb{N}$ is a natural number and $\mathbf{0} \in \mathbb{R}^{n\times d}$ is a matrix with all zero entries.

Proof The proof follows the same idea as Corollary 3, but the SVD of $\tilde{F}$ is simpler than Corollary 3. If the SVD of F is $F = U\Sigma V^T$, then the SVD of $\tilde{F} = [F, \mathbf{0}]$ is $\tilde{F} = \tilde{U}\tilde{\Sigma}\tilde{V}^T$, where $\tilde{U} = U, \tilde{\Sigma} = [\Sigma, \mathbf{0}], \tilde{V} = \begin{bmatrix} V & \\ & W \end{bmatrix}$, with $W \in \mathbb{R}^{d\times d}$ an orthogonal matrix that satisfies $W^TW = I_d$. Note that $\tilde{\Sigma} = [\Sigma, \mathbf{0}]$ translates into $\tilde{\sigma}_i^2 = \begin{cases} \sigma_i^2 & 1 \le i \le D \\ 0 & D+1 \le i \le D+d \end{cases}$.

Iterative invariant: if we apply Algorithm 2 to both $\tilde{F}$ and F, with the same initialization $\tilde{\alpha} = 1, \tilde{\beta} = 1$, then $\tilde{\alpha} = \alpha, \tilde{\beta} = \beta$ holds before Line 5. Suppose $\tilde{\alpha} = \alpha, \tilde{\beta} = \beta$ holds before a while-loop, then we have:

$$\tilde{\gamma} = \sum_{i=1}^{D+d} \frac{\tilde{\beta}\tilde{\sigma}_i^2}{\tilde{\alpha} + \tilde{\beta}\tilde{\sigma}_i^2} = \sum_{i=1}^{D} \frac{\beta\sigma_i^2}{\alpha + \beta\sigma_i^2} = \gamma$$

$$\tilde{\Lambda} = \operatorname{diag}\left\{\tilde{\alpha} + \tilde{\beta}\tilde{\sigma}_i^2\right\}, \tilde{\alpha} + \tilde{\beta}\tilde{\sigma}_i^2 = \begin{cases} \alpha + \beta\sigma_i^2 & 1 \le i \le D \\ \alpha & D+1 \le i \le D+d \end{cases}$$

$$\tilde{m} = \tilde{\beta}\tilde{A}^{-1}\tilde{F}^T y = \beta \begin{bmatrix} V & \\ & W \end{bmatrix} \begin{bmatrix} \Lambda^{-1} & \\ & \frac{1}{\alpha} I_d \end{bmatrix} \begin{bmatrix} V^T & \\ & W^T \end{bmatrix} \begin{bmatrix} F^T \\ \mathbf{0}_{n\times d}^T \end{bmatrix} y = \begin{bmatrix} m \\ \mathbf{0}_{d\times 1} \end{bmatrix}$$

$$\tilde{m}^T\tilde{m} = m^T m, \tilde{F}\tilde{m} = [F, \mathbf{0}_{n\times d}] \begin{bmatrix} m \\ \mathbf{0}_{d\times 1} \end{bmatrix} = Fm.$$

After the while-loop iteration, $\tilde{\alpha}' = \frac{\tilde{\gamma}}{\tilde{m}^T\tilde{m}} = \frac{\gamma}{m^T m} = \alpha', \tilde{\beta}' = \frac{n-\tilde{\gamma}}{||\tilde{F}\tilde{m}-y||_2^2} = \frac{n-\gamma}{||Fm-y||_2^2} = \beta'$, then the iterative invariant $\tilde{\alpha} = \alpha, \tilde{\beta} = \beta$ still holds. Therefore, we know that when the algorithm converges, $\tilde{\alpha}^* = \alpha^*, \tilde{\beta}^* = \beta^*$. The corresponding maximum evidence is

$$\begin{aligned}
\tilde{\mathcal{L}} &= \frac{n}{2}\log\tilde{\beta}^* + \frac{D+d}{2}\log\tilde{\alpha}^* - \frac{n}{2}\log 2\pi - \frac{\tilde{\beta}^*}{2}||\tilde{F}\tilde{m}-y||_2^2 - \frac{\tilde{\alpha}^*}{2}\tilde{m}^T\tilde{m} - \frac{1}{2}\log\left|\tilde{A}^*\right| \\
&= \frac{n}{2}\log\beta^* + \frac{D+d}{2}\log\alpha^* - \frac{n}{2}\log 2\pi - \frac{\beta^*}{2}||Fm-y||_2^2 - \frac{\alpha^*}{2}m^T m - \frac{1}{2}\log\left|\tilde{\Lambda}^*\right| \\
&= \frac{n}{2}\log\beta^* + \frac{D+d}{2}\log\alpha^* - \frac{n}{2}\log 2\pi - \frac{\beta^*}{2}||Fm-y||_2^2 - \frac{\alpha^*}{2}m^T m \\
&\quad - \frac{1}{2}\log|\Lambda^*| - \frac{1}{2}\log(\alpha^*)^d \\
&= \mathcal{L} + \frac{d}{2}\log\alpha^* - \frac{d}{2}\log\alpha^* \\
&= \mathcal{L}.
\end{aligned}$$

■

Appendix F. Detailed Descriptions of the Datasets

Aircraft: The dataset contains fine-grained classification of 10,000 aircraft pictures which belong to 100 classes, with 100 images per class.

Birdsnap: The dataset contains 49,829 images of 500 species of North American birds.

Caltech: The dataset contains 9,144 pictures of objects belonging to 101 categories. There are about 40 to 800 images per category. Most categories have about 50 images.

Cars: The dataset contains 16,185 images of 196 classes of cars. The data is split into 8,144 training images and 8,041 testing images.

CIFAR 10: The dataset consists of 60,000 32×32 colorful images in 10 classes, with 6,000 images per class. There are 50,000 training images and 10,000 test images.

CIFAR 100: The dataset is just like the CIFAR 10, except it has 100 classes containing 600 images each.

DTD: The dataset contains a collection of 5,640 textural images in the wild, annotated with a series of human-centric attributes. It has 47 classes and 120 images per class.

Pets: The dataset contains 7,049 images of cat and dog species which belong to 47 classes, with around 200 images per class.

SUN: The dataset contains 39,700 scenery pictures with 397 classes and 100 samples per class.

Appendix G. Original Results in Figures

Original results in figures are shown in the Table 10, Table 11, and Table 12.

Table 10: Original results in Figure 4.

task		ResNet-34	ResNet-50	ResNet-101	ResNet-152	WideResNet-50	DenseNet-121	DenseNet-169	DenseNet-201	Inception v1	Inception v3	MobileNet v2	NASNet-A Mobile	τ_w
Aircraft	Accuracy	79.9	86.6	85.6	85.3	83.2	85.4	84.5	84.6	82.7	88.8	82.8	72.8	-
	Laplace	-2.864	-3.127	-3.080	-3.158	-3.721	-2.235	-1.906	-1.754	-2.382	-2.822	-2.217	-1.481	**-0.32**
	LEEP	-0.497	-0.412	-0.349	-0.308	-0.337	-0.431	-0.340	-0.462	-0.795	-0.492	-0.515	-0.506	0.13
	NCE	-0.364	-0.297	-0.244	-0.214	-0.248	-0.296	-0.259	-0.322	-0.348	-0.250	-0.411	-0.444	0.39
	LogME	0.930	0.946	0.948	0.950	0.934	0.938	0.943	0.942	0.934	0.953	0.941	0.948	**0.59**
Birdsnap	Accuracy	59.5	74.7	73.8	74.3	63.1	73.2	71.4	72.6	73.0	77.2	69.3	68.3	-
	LEEP	-1.758	-1.647	-1.553	-1.481	-1.554	-1.729	-1.756	-1.645	-2.483	-1.776	-1.951	-1.835	0.19
	NCE	-1.640	-1.538	-1.479	-1.417	-1.399	-1.566	-1.644	-1.493	-1.807	-1.354	-1.815	-1.778	0.51
	LogME	0.802	0.829	0.836	0.839	0.825	0.810	0.815	0.822	0.806	0.848	0.808	0.824	**0.66**
Caltech	Accuracy	90.2	91.8	93.1	93.2	91.0	91.9	92.5	93.4	91.7	94.3	89.1	91.5	-
	LEEP	-2.249	-2.195	-2.067	-1.984	-2.179	-2.159	-2.039	-2.122	-2.718	-2.286	-2.373	-2.263	0.30
	NCE	-1.899	-1.820	-1.777	-1.721	-1.828	-1.807	-1.774	-1.808	-1.849	-1.722	-2.009	-1.966	**0.69**
	LogME	1.362	1.509	1.548	1.567	1.505	1.365	1.417	1.428	1.440	1.605	1.365	1.389	0.66
Cars	Accuracy	86.4	91.7	91.7	92.0	89.7	91.5	91.5	91.0	91.0	92.3	91.0	88.5	-
	LEEP	-1.534	-1.570	-1.370	-1.334	-1.406	-1.562	-1.505	-1.687	-2.149	-1.637	-1.695	-1.588	0.26
	NCE	-1.203	-1.181	-1.142	-1.128	-1.183	-1.111	-1.192	-1.319	-1.201	-1.195	-1.312	-1.334	0.36
	LogME	1.245	1.253	1.255	1.260	1.250	1.249	1.252	1.251	1.246	1.259	1.250	1.254	**0.69**
CIFAR10	Accuracy	97.1	96.8	97.7	97.9	97.7	97.2	97.4	97.4	96.2	97.5	95.7	96.8	-
	LEEP	-3.418	-3.407	-3.184	-3.020	-3.335	-3.651	-3.345	-3.458	-4.074	-3.976	-3.624	-3.467	0.72
	NCE	-3.398	-3.395	-3.232	-3.084	-3.348	-3.541	-3.427	-3.467	-3.338	-3.625	-3.511	-3.436	0.51
	LogME	0.323	0.388	0.463	0.469	0.398	0.302	0.343	0.369	0.293	0.349	0.291	0.304	**0.82**
CIFAR100	Accuracy	84.5	84.5	87.0	87.6	86.4,	84.8	85.0	86.0	83.2	86.6	80.8	83.9	-
	LEEP	-3.531	-3.520	-3.330	-3.167	-3.391	-3.715	-3.525	-3.643	-4.279	-4.100	-3.733	-3.560	0.66
	NCE	-3.230	-3.241	-3.112	-2.980	-3.158	-3.304	-3.313	-3.323	-3.253	-3.447	-3.336	-3.254	0.53
	LogME	1.036	1.099	1.130	1.133	1.102	1.029	1.051	1.061	1.037	1.070	1.039	1.051	**0.77**
DTD	Accuracy	70.0	75.2	76.2	75.4	70.1	74.9	74.8	74.5	73.6	77.2	72.9	72.8	-
	LEEP	-3.670	-3.663	-3.718	-3.653	-3.764	-3.847	-3.646	-3.757	-4.124	-4.096	-3.805	-3.691	-0.06
	NCE	-3.104	-3.119	-3.199	-3.138	-3.259	-3.198	-3.218	-3.203	-3.082	-3.261	-3.176	-3.149	-0.35
	LogME	0.704	0.761	0.757	0.766	0.731	0.710	0.730	0.730	0.727	0.746	0.712	0.724	**0.50**
Pets	Accuracy	92.3	92.5	94.0	94.5	92.8	92.9	93.1	92.8	91.9	93.5	90.5	89.4	-
	LEEP	-1.174	-1.031	-0.915	-0.892	-0.945	-1.100	-1.111	-1.108	-1.520	-1.129	-1.228	-1.150	0.66
	NCE	-1.094	-0.956	-0.885	-0.862	-0.900	-0.987	-1.072	-1.026	-1.076	-0.893	-1.156	-1.146	**0.83**
	LogME	0.835	1.029	1.061	1.084	1.016	0.839	0.874	0.908	0.913	1.191	0.821	0.833	0.61
SUN	Accuracy	63.1	64.7	64.8	66.0	67.4	62.3	63.0	64.7	62.0	65.7	60.5	60.7	-
	LEEP	-2.727	-2.611	-2.531	-2.513	-2.569	-2.713	-2.570	-2.618	-3.153	-2.943	-2.764	-2.687	0.54
	NCE	-2.573	-2.469	-2.455	-2.444	-2.457	-2.500	-2.480	-2.465	-2.534	-2.529	-2.590	-2.586	0.68
	LogME	1.704	1.744	1.749	1.755	1.750	1.704	1.716	1.718	1.715	1.753	1.713	1.721	**0.71**

Table 11: Original results in Figure 5.

task		ResNet-34	ResNet-50	ResNet-101	ResNet-152	WideResNet-50	DenseNet-121	DenseNet-169	DenseNet-201	Inception v1	Inception v3	MobileNet v2	NASNet-A Mobile	τ_w
dSprites	MSE	0.037	0.031	0.028	0.028	0.034	0.039	0.035	0.036	0.045	0.044	0.037	0.035	-
	LogME	1.05	1.53	1.64	1.63	1.31	1.35	1.25	1.34	1.18	1.22	1.18	1.39	0.79

Table 12: Original results in Figure 9. (Popularity is measured by download count in millions.)

task		RoBERTa	RoBERTa-D	uncased BERT-D	cased BERT-D	ALBERT-v1	ALBERT-v2	ELECTRA-base	ELECTRA-small	τ_w
MNLI	Accuracy	87.6	84.0	82.2	81.5	81.6	84.6	79.7	85.8	-
	LogME	-0.568	-0.599	-0.603	-0.612	-0.614	-0.594	-0.666	-0.621	0.66
	Popularity	3.78	0.61	6.01	1.09	0.11	1.25	0.13	0.23	0.28
QQP	Accuracy	91.9	89.4	88.5	87.8	-	-	-	-	-
	LogME	-0.465	-0.492	-0.488	-0.521	-	-	-	-	0.73
	Popularity	3.78	0.61	6.01	1.09	-	-	-	-	0.00
QNLI	Accuracy	92.8	90.8	89.2	88.2	-	-	-	-	-
	LogME	-0.565	-0.603	-0.613	-0.618	-	-	-	-	1.00
	Popularity	3.78	0.61	6.01	1.09	-	-	-	-	0.00
SST-2	Accuracy	94.8	92.5	91.3	90.4	90.3	92.9	-	-	-
	LogME	-0.312	-0.330	-0.331	-0.353	-0.525	-0.447	-	-	0.68
	Popularity	3.78	0.61	6.01	1.09	0.11	1.25	-	-	0.38
CoLA	Accuracy	63.6	59.3	51.3	47.2	-	-	-	-	-
	LogME	-0.499	-0.536	-0.568	-0.572	-	-	-	-	1.00
	Popularity	3.78	0.61	6.01	1.09	-	-	-	-	0.00
MRPC	Accuracy	90.2	86.6	87.5	85.6	-	-	-	-	-
	LogME	-0.573	-0.586	-0.605	-0.604	-	-	-	-	0.53
	Popularity	3.78	0.61	6.01	1.09	-	-	-	-	0.33
RTE	Accuracy	78.7	67.9	59.9	60.6	-	-	-	-	-
	LogME	-0.709	-0.723	-0.725	-0.725	-	-	-	-	1.00
	Popularity	3.78	0.61	6.01	1.09	-	-	-	-	-0.29

Appendix H. Complete Results in Prompt Learning

Table 13: Complete results in prompt learning with manually selected anchor words.

accuracy (anchor P \ anchor N)	negative	bad	ill	evil	poor
positive	49.8	52.8	49.1	49.1	60.9
good	51.0	50.9	49.0	52.4	50.9
fine	51.7	51.0	49.1	54.5	50.9
great	55.4	53.1	49.1	**61.4**	51.0
nice	51.6	50.6	49.1	51.0	50.8

Appendix I. Full Figure in Convergence Analysis

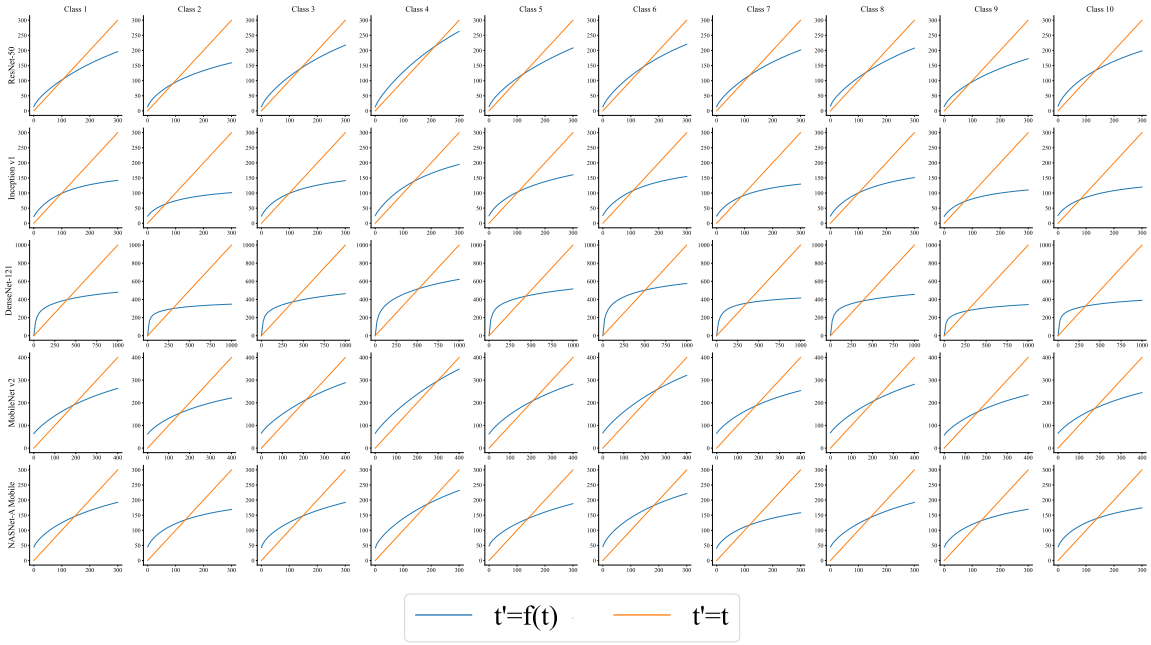


Figure 13: Fixed points of $f(t)$ in Equation 3 for all 10 classes in CIFAR10 with 5 pre-trained models. We plot $t' = f(t)$ (in blue) and $t' = t$ (in orange), whose intersections are fixed points. The existence of fixed points guarantees the convergence of the evidence maximization procedure in LogME.