Model Selection with Model Zoo via Graph Learning

Ziyu Li Hilco van der Wilk Danning Zhan Megha Khosla
Asterios Katsifodimos Alessandro Bozzon Rihan Hai
Delft University of Technology
z.li-14,{initials.lastname}@tudelft.nl

***Abstract*—Pre-trained deep learning (DL) models are increasingly accessible in public repositories, i.e., model zoos. Given a new prediction task, finding the best model to fine-tune can be computationally intensive and costly, especially when the number of pre-trained models is large. Selecting the right pre-trained models is crucial, yet complicated by the diversity of models from various model families (like ResNet, Vit, Swin) and the hidden relationships between models and datasets. Existing methods, which utilize basic information from models and datasets to compute scores indicating model performance on target datasets, overlook the intrinsic relationships, limiting their effectiveness in model selection. In this study, we introduce TransferGraph, a novel framework that reformulates model selection as a graph learning problem. TransferGraph constructs a graph using extensive metadata extracted from models and datasets, while capturing their inherent relationships. Through comprehensive experiments across 16 real datasets, both images and texts, we demonstrate TransferGraph's effectiveness in capturing essential model-dataset relationships, yielding up to a 32% improvement in correlation between predicted performance and the actual fine-tuning results compared to the state-of-the-art methods.**

I. Introduction

Deep learning has been widely used in handling unstructured data, including tasks related to image and text classification. The paradigm of *first pre-training, then fine-tuning* has become the de facto of applying deep learning in practice. Pre-training is the phase of training a neural network on a large, diverse dataset, typically drawn from a general domain, e.g., ImageNet [1]. Subsequently, the fine-tuning step refines the model for a specific task, often a smaller target dataset. This two-step process leverages the general knowledge acquired during pre-training, facilitating effective adaptation to a narrower and more specialized context. The general representations learned during pre-training, speed up model convergence during fine-tuning and help reduce the risk of over-fitting.

Today, many pre-trained models are available in public online platforms, e.g., HuggingFace[1], TensorFlow Hub[2], and PyTorch Hub[3]. Such repositories of pre-trained models are referred to as *model zoos*. Model zoos have been widely adopted in recent years, as they offer convenient access to a collection of pre-trained models, including cutting-edge deep learning architectures. This lowers the expertise barrier, enabling non-expert individuals to apply complex deep learning models in their applications. Utilizing a model zoo for fine-tuning facilitates the adaptation across a wide range of target datasets, which have varying quantities of training data [2]. In addition, by fine-tuning pre-trained models from the model zoo, machine learning practitioners can bypass the need for training from scratch—a resource-intensive process, resulting in significant savings in both development time and computational resources.

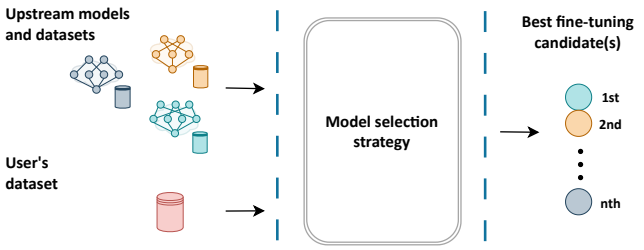


Figure 1: Illustration of the model selection problem setting.

However, it is a non-trivial task to pick the *right* pre-trained models as the starting point of fine-tuning, which has a substantial impact on the effectiveness of the fine-tuning results [2]. A straightforward solution is to fine-tune all the relevant pre-trained models, which is computationally expensive, and sometimes infeasible in practice. For instance, there are 7411 models for image classification tasks in the HuggingFace repository and 900 variations on TensorFlow Hub. It took 1178 hours of GPU time to fine-tune all the 185 models in our model zoo on a single dataset.

The practical choice is to identify pre-trained models that exhibit promising performance even without fine-tuning, i.e., *model selection*. As in Figure 1, given a target dataset and several pre-trained models over existing datasets, model selection aims to rank and select optimal candidates from the model zoo to perform fine-tuning. Different strategies may yield disparate rankings of the candidates.

A naive approach is to randomly select models for fine-tuning. This random selection strategy may suffice when pre-trained models all have similar fine-tuning accuracy. However, in the more general case where the performance of models varies, the random strategy is ineffective. In Figure 2, we report the results of the average accuracy of the top five models

[1]https://huggingface.co/
[2]https://www.tensorflow.org/
[3]https://pytorch.org/hub/

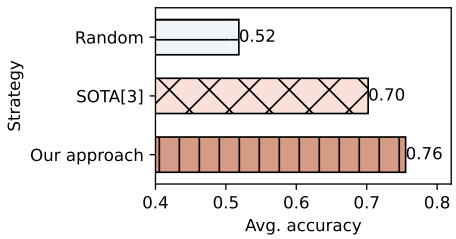


Figure 2: Average fine-tuned accuracy of the top 5 selected models compared between random selection strategy and our proposed solution learning from a graph along with metadata (example dataset: stanfordcars).

selected through diverse strategies (full results in Section VII). `Random` denotes a random selection strategy, which only achieved an unsatisfying accuracy value of 0.52.

Existing studies [3]–[7] mainly focus on extracting information about the pre-trained models and datasets, and mapping model features to the target dataset labels to measure the model *transferability*. The efficacy of features is expected to diminish as the source dataset (training dataset of the pre-trained model) and target dataset become less similar [8]. Another approach, exemplified by Amazon LR [9], learns the pattern of model performance by using metadata (e.g., model architecture, data size) to train a regression model. The mechanism of the previous methods is limited to applying model representations extracted from the learned parameters or features constructed from metadata, overlooking the deeper connections inherent among models and datasets.

Our work advances beyond existing studies by incorporating the prior knowledge of fine-tuning and transferability scores (e.g., LogME [3]), representing this information through weighted connections between models and datasets. We borrow the inspiration from data management systems for data repositories, such as data lakes [10]–[12]. For managing a collection of datasets, a common approach is to structure these datasets as graphs [13]–[15]. This involves representing tables as nodes and their relationships as edges. For instance, an edge can indicate that two tables are semantically similar [13]. For the model selection problem, rich relationships exist not only between models and datasets, but also among datasets themselves. Our approach leverages the additional information on relationships informed by fine-tuning and transferability scores, and dataset similarity.

We reformulate the challenge of model selection as a graph link prediction problem. We propose *TransferGraph*[4], which explores how the relationships among *dataset-dataset* and *dataset-model* can facilitate more effective model selection, offering a structured and intuitive method to navigate and understand these complex relationships. To represent and analyze these intricate relationships, we represent them using graph structures. We show that TransferGraph is able to identify suitable pre-trained models for the target dataset by exploiting graph features learned from the graph structure and along with other metadata information (e.g., model architecture, data size). As shown in Figure 2, TransferGraph outperforms the state-of-the-art method [3] with a notable improvement in fine-tuning accuracy.

[4]Code is available at https://github.com/TransferGraph/transfergraph and HuggingFace organization at https://huggingface.co/TransferGraph

Contributions. We summarize our contributions as follows.

- We reformulate the model selection problem as a graph learning problem.
- Different from existing works, which only take into account the dataset labels or solely information about models or datasets, we exploit the metadata of both models and datasets and further learn the inherent relationships between the artifacts by learning a graph.
- We propose a framework that tackles the model selection problem via graph learning. The framework consists of end-to-end processes, from feature collection and graph learning to model performance prediction.
- Extensive experiments are conducted to evaluate the validity of our graph-based model selection strategy. We show that with a graph learning method, we can predict model performance with a high correlation to the actual fine-tuning accuracy for both image and text classification tasks.

II. Background and problem definitions

It is a challenging task to select the *right* models for fine-tuning, especially given the abundant pre-trained models in the model zoo. In this section, we explain existing model selection strategies and identify their limitations.

A. Model selection strategies

Previous model selection strategies consider features of models and datasets from different perspectives. They mainly differ in the information of datasets and pre-trained models they use. Some works like SHiFt [16] have developed systems which combine these approaches, while also taking user inputs such as budget constraints into account. We categorize these model selection strategies based on the features they employ to rank the models for model selection.

Task-similarity-based model selection. Early model selection methods use the similarity of the source and target tasks to measure the transferability of a model. When the target task is similar to source task, a model with good performance on the source task is likely to have good fine-tuning performance [17]. Methods in this group include EMD [18] and NCE [4], Task2Vec [4], [19]. To obtain the similarity of the source and target tasks, EMD [18] and NCE [4] compare source and target task features and labels. Task2Vec [19] embeds tasks as vectors using a single probe model and computes their pairwise distances as transferability scores.

Feature-based model selection. More recent approaches leverage the target task specific features, which are extracted by executing a forward pass of the target task on each pre-trained model (model inference on the target dataset). Methods

in this group include LEEP [5], LogME [3], PARC [6] and TransRate [7]. These approaches circumvent the need for fine-tuning. However, as the number of pre-trained models in a model zoo grows, it becomes inefficient to perform a forward pass over all pre-trained models, even infeasible. Moreover, methods in this group overlook basic features of both the target dataset (like the number of samples and labels) and the pre-trained model (such as input size and architecture), which are crucial for fine-tuning efficiency. For instance, a mismatch in input size or the number of classes, leads to significant deterioration in fine-tuning performance [9].

Learning-based model selection. The third group of approaches [6], [9] trains a simple linear model, e.g., a linear regression model, to predict model performance and recommend pre-trained models given a new target task. The used features extracted from models or *metadata* of the target dataset and pre-trained models. The state-of-the-art approach, Amazon LR [9], employs only basic metadata of the target dataset and pre-trained models. It achieved competitive results when learning a linear regression model. The authors suggest that incorporating additional features could further improve this method.

B. Limitations and Challenges

We summarize the limitations of existing model selection strategies and outline the challenges to tackle them.

1) Overlooking the heterogeneity of model zoo: A model zoo may encompass heterogeneous models and datasets. Models within this context can exhibit differences in the pre-trained domain, architecture, and hyperparameter settings. At the same time, datasets vary in terms of the tasks they address and the distribution of their data. Predicting the performance and capability of models is challenging, given that the models are trained differently, and the inductive biases of models are different. Featured-based model selection strategies usually use the model as feature extractor, or assumes the fine-tuning process does not change the backbone weights much [6], [20]. However, such an assumption does not hold in practice.

Prior studies [7], [20] have often restricted model architectures to certain categories, e.g., ResNet, MobileNet or DenseNet. In LogMe [3], only models pre-training on the same source dataset (e.g., ImageNet) are included. However, the optimal architecture or Pareto-optimal models are usually task-dependent, relying on the inductive bias of the model and the dataset properties [9]. Fine-tuning with a model zoo helps transfer to a diverse set of target tasks with different downstream datasets. Due to the diversity of the model characteristics, e.g., architecture family, pre-trained domain, and hyperparameter settings, it is even more challenging to identify suitable candidate models for the downstream task.

2) Insufficient feature coverage: Feature-based model selection strategies [6], [20] often rely on the model features and the target dataset labels. They assume that models can generalize better if the features extracted by the model are similar and labels are similar. However, such approaches struggle to accurately predict top-performing models for target datasets significantly different from source datasets used for pre-training [9]. Conversely, incorporating simple prior knowledge, such as dataset characteristics, is proven to help predict the model performance on downstream datasets [9]. Learning from basic metadata of models and datasets is beneficial yet limited due to its coarse-grained nature. It often overlooks the intricate relationships between models and datasets. Therefore, a central challenge lies in identifying and utilizing such inherent relationships for more effective model selection.

III. Model selection as a graph learning problem

We first explain the problem setting of model selection. To tackle the challenges in Sec. II-B, we propose transforming the model selection problem into a graph learning problem.

A. Problem definition

Consider a set of models, denoted as $M = \{m_1, ...m_N\}$, and a collection of datasets, represented as $\chi = \{d_1, ...d_K\}$. We denote the actual fine-tuning accuracy as $T_{i,j}$, with respect to the model m_i and the target dataset d_j, where $m_i \in M$, and $d_j \in \chi$. Given a pre-trained model m_i, we are interested in predicting a score S_{ij} which approximates its fine-tuning accuracy on the target dataset d_j.

The predicted score should be a good approximation of the actual fine-tuning results and exhibit a strong correlation with the target dataset. Such an alignment would enable the predicted score to be a reliable indicator of fine-tuning performance on the target dataset, allowing for the effective ranking of pre-trained models.

To measure the effectiveness of the model selection score, we use *Pearson's correlation coefficient*, following the common practice [9]. We use $\tau \in [-1, 1]$ to represent the Pearson's correlation. Given N paired data $\{(m_1, d_t), (m_2, d_t), ...(m_N, d_t)\}$, τ is defined as:

$$\tau = \frac{\sum_{i=1}^{N}(T_i - \bar{T})(S_i - \bar{S})}{\sqrt{\sum_{i=1}^{N}(T_i - \bar{T})^2 \sum_{i=1}^{N}(S_i - \bar{S})^2}} \tag{1}$$

The goal is to maximize the correlation between the predicted scores and the model performance. An absolute value of 1 implies that a S perfectly aligns with the trend of T with all data points lying on a line.

B. Convert model selection to graph learning

In this work, we formulate the model selection problem as a graph learning process, which maximizes the Pearson correlation τ, between the predicted score S_{ij} and the actual fine-tuning accuracy T_{ij}.

Definition III.1 (Graph)**.** We denote a graph as $G = (V, E)$ where V is the set of vertices and $E \subseteq V \times V$ denotes the set of edges that connect the vertices in V.

In our setting, a vertex either represents a dataset or a model. Given a set of datasets $\chi = \{d_1, d_2, ..., d_K\}$, and a set of pre-trained deep learning models $M = \{m_1, m_2, ..., m_N\}$, we build our vertex set as $V = \chi \cup M$. Here $K = |\chi|$ is the number of datasets, and $N = |M|$ is the number of models.

Table I: Notation definitions

Notation	Definition
$S_{i,j}$	Predicted transferability score of m_i on d_j
$T_{i,j}$	Fine-tuning performance of m_i on d_j
χ, d_j	Set of datasets collection and dataset j
M, m_i	Model collection and model i
ϕ	Dataset similarity
G	Graph
E	Edge of Graph
V	Vertex / Node
L	Edge labels
τ	Pearson's Correlation
$f_G()$	Function over the graph
$W^{(k)}$	Weights of the graph
$Q^{(k)}$	Operation that allows aggregation afterwards
$\bar{X}$	Mean of the variable X
$\hat{X}$	Prediction for the variable X
$F()$	Learned function to predict the model performance

In our graph, we construct three types of edges, depending on the nodes connected by the edges. The first type links dataset nodes, utilizing calculated dataset similarity for connection. The second type forms connections between a model and a dataset, representing existing transferability scores, e.g., LogME [3], PARC [6]. The third edge type also connects a model and a dataset, and comes from the training history of each model on each dataset, such as the pre-trained performance and fine-tuning performance. In the graph, instead of having the binary adjacency matrix, the respective scores will be used as the weights of the adjacency matrix. Instead of having a fully connected graph, a pruning threshold will be used to decide the existence of the edges.

In this work, we are interested in exploiting the inherent relationships between models and datasets for the model selection problem in a model zoo. Borrowing the concept from data lake management, we represent the model and dataset relationships in a graph and learn the graph structure by performing a link prediction task.

Link prediction. We extend Definition III.1 to $G = (V, E, L)$, by adding the set of labels or representations of each edge, denoted as L. The goal of link prediction is to learn a predictive model that assigns a score to pairs of vertices (u, v), indicating the likelihood of an edge existing between them.

In our problem setting, we aim to identify the models that have high performance on the datasets. We can specify the positive edges with models receiving high performance in the training history. We regard the link prediction task as a classification problem, positive edges as 1 and negative edges as 0. Thus edges of models performing well on a dataset have the label of 1 and 0 if the model has poor performance on a dataset, which forms the labels in L. We use sigmoid to generate output and compute the loss, which results in probability scores within [0,1].

We formulate the model selection problem through a learned function over the graph, represented by the following formula.

$$\hat{T}_{i,j} = F(f_G(m_i), f_G(d_j)), \tag{2}$$

In Equation 2, we use the graph learners f_G to learn the set

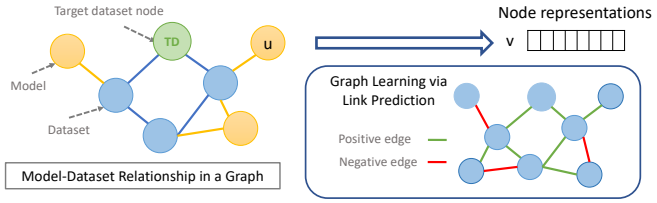


Figure 3: Link prediction in the context of model selection

of labels L for the link prediction task on our constructed graph. $f_G(m_i)$ obtains the vertex embeddings of m_i, and $f_G(d_j)$ obtaining the vertex embeddings of d_j. F denotes the prediction model that maps from the model and data representations to the fine-tuning results. The prediction model is trained on the training history.

We reformulate the model selection problem with a model zoo as a graph link prediction problem. In what follows, we will introduce the information needed to tackle the problem in our proposed graph-learning-based strategy.

IV. Data collection: Metadata and Features

Extensive research [3], [5], [6] has been conducted to investigate the relationship between the model features and the target dataset labels. Yet, the metadata of models and datasets are often neglected. Though simple and coarse-grained, such metadata are of great value to specify the characteristics of the models and datasets in some sense, and prove to be useful for predicting the fine-tuning performance [9]. Below, we will introduce the main metadata and features considered.

A. Metadata as features

In the following subsections, we present the considered metadata of both models and datasets.

1) Metadata of datasets: The metadata of datasets can be indicators of the fine-tuning difficulty. The properties of a dataset can affect a model's performance. For example, a dataset with many classes is more difficult to learn than a dataset with binary classes. We do not exclude the information from the pre-trained model, as in most feature-based model selection strategies. We consider the metadata of both source and target datasets for model selection.

Number of data samples. A small dataset contains less information and is likely easier to learn. In contrast, a large dataset with more diverse features may require a more complex model to learn to obtain good performance.

Number of label classes. A multi-label classification problem is more challenging than binary classification and may require more data samples to learn.

2) Metadata of models: The metadata of models reveals their learning capability from a certain perspective. A model with more parameters may capture more generalized features. Models with different architectures may have varying inductive biases for different datasets.

Input shape. More information can be captured with a larger input shape, e.g., a higher-resolution image.

Architecture. The architecture of a model plays an important role in determining how well a model can capture complex patterns in a dataset. A more complex architecture, e.g., ResNet [21], Inception [22], might be more suitable to learn more complex and larger inputs than e.g., LeNet [23].

Pre-trained dataset. The source data quality significantly impacts the learned features and knowledge that a model can capture. A model trained on a large dataset with diverse data may have more generalized ability than one trained on a small and biased dataset.

Model performance. The performance identifies the capability of a model. For example, when two models are trained on the same dataset, the model with higher accuracy indicates that it has better knowledge of the dataset and may be adaptable to new datasets.

Number of parameters. A bigger model with more parameters can capture more generalized features from a large dataset. Compared to an SVM model, a more complex model (ResNet) can perform better in image classification on ImageNet.

Memory consumption. The memory consumption of a model is correlated to the number of parameters. It is another indicator of the complexity of a model.

This work does not include all the metadata mentioned in Amazon LR [9]. Some metadata included in Amazon LR needs further computation to obtain, e.g., dataset difficulty. The metadata mentioned above are more accessible to obtain. In addition, we include some other features, e.g., models' pre-trained performance compared to Amazon LR. We find that even with the simple metadata, the model selection strategies can make good predictions on the model performance.

B. Dataset Features

Together with the metadata, we capture the dataset features in the feature collection stage. Similar to feature-based model selection strategies, which acquire features by executing a forward pass over all candidate models on the target dataset, we can capture dataset representations through a comparable approach. By utilizing a reference ML model, referred to as a *probe network*, for inference on datasets as the initial step. We acknowledge that the probe network exhibits varied performance on different datasets, resulting in distinct embeddings within a vector space. We expect these embeddings to unveil the distinctive characteristics of the datasets, and the distance between embeddings captures the semantic similarities between the datasets.

1) Dataset representations: Prior studies, including Domain Similarity [18], Task2Vec [19] and Taskonomy [24], focus on learning dataset representations within the realm of transfer learning. We adopt *Domain Similarity* to extract the embeddings for dataset representations.

Domain Similarity embeddings. We adopt a similar mechanism to extract features of data samples from large pre-trained model as in Domain Similarity [18]. We aggregate all the representations of the dataset inferred by a probe network as the dataset features. The probe network is usually a large network, such as VGG [25], ResNet [21]. These networks are pre-trained on large corpus of data, e.g., ImageNet [1] for images or millions of collected texts, and are considered to be able to capture good generic features from the images and thus serve as reference models to retrieve features. The embedding of a dataset d_k is defined as:

$$\tilde{E}_k = \sum_{j=1}^{|d_k|} g(x_j), \quad x_j \in d_k, \tag{3}$$

$g(\cdot)$ represents the features obtained by extracting the feature layers of the reference model. We adopt *ResNet34* pretrained on ImageNet as the reference model for image datasets and *GPT-Neo* [26] for textual datasets.

2) Dataset similarity: A model with good performance on the source task is likely to have good fine-tuning performance when the target task is similar [17]. The similarity between datasets is denoted by ϕ. This similarity is quantified by calculating the correlation distance between datasets, where a shorter distance signifies greater similarity. We expect a higher similarity between semantically similar datasets. For example, a dataset of flowers shall be more similar to a dataset of plants than airplanes.

C. Other features

Existing works such as Model2Vec [19] and attribution map [27] have investigated to obtain model features for transfer learning. Future work can investigate using model features as an additional type of feature for predicting the model performance.

V. Graph Construction and Learning

The metadata and dataset features mentioned in Section IV characterize the datasets and models from a high-level perspective. When the metadata information and dataset features are similar, distinguishing between them becomes challenging, leading to difficulties in predicting model performance. In order to obtain more subtle features of models and datasets, we aim to explore the intrinsic relationships between models and datasets. For example, whether a model's proficiency on one dataset implies good performance on a similar dataset, or whether models pre-trained on diverse datasets exhibit distinct performance on a given target dataset.

We introduce a graph-based approach to capture and leverage the relationships between models and datasets. Specifically, we explicitly incorporate prior knowledge about dataset-dataset and model-dataset as edges/relationships in the graph. Through graph-based learning, we seek to exploit not only the available node features but also the inherent assumptions or preferences (inductive biases) embedded within the topology of the graph. The subsequent section will detail how we construct this graph, tailored for model selection with a model zoo. In addition, we introduce the representative graph learning algorithms that capture the information of the constructed graph.

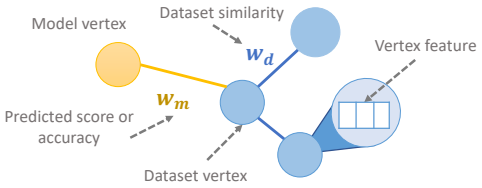


Figure 4: Graph properties

A. Graph construction

To assign attributes to nodes and edges, it is crucial to identify entities and relationships. In Figure 4, we present an overview of the graph structure, where vertices and edges may carry distinct semantic meanings.

1) Vertices: A vertex in the constructed graph can be either a model or a dataset. The vertices are connected to each other, embedded with model-dataset relationships or dataset-dataset relationships. Usually, model zoos contain models trained on overlapping publicly available (benchmark) datasets, making the number of models exceed the number of datasets in a model zoo.

2) Vertex features: A vertex can be embedded with features. Some graph learners, e.g., GraphSAGE [28], GAT [29], can capture the vertex features and use them to initiate the learning process. We introduce dataset features earlier in Section IV-A1. We can embed the dataset features as the features of the dataset vertices.

3) Edges and edge attributes: The edges are constructed in three ways: i) edges between datasets indicating the similarity between datasets, ii) model performance on datasets as edges between models and datasets, iii) predicted scores obtained from other feature-based model selection strategies as another type of edges between models and datasets.

Dataset-Dataset (D-D) edge attributes. The construction of D-D edges is achieved by evaluating the similarity of dataset representations. The dataset similarity is denoted as ϕ. The computation encompasses all possible dataset pairs, with the resulting similarity scores employed as edge attributes.

Model-Dataset (M-D) edge attributes. The edges between datasets and models are associated with different meanings. A model can connect to a dataset with training performance or predicted score. For example, if we can obtain the pre-trained performance of $m_{Resnet50}$ on the dataset *cifar100* with an accuracy of, e.g., 95%, the vertices between $m_{Resnet50}$ and *cifar100* has an edge with an attribute of 0.95. We can also embed the fine-tuning results if they are available. In addition, the predicted scores obtained from other model selection strategies can also embed meaningful information between models and datasets.

B. Graph Learning

In the context of model selection, we formulate the graph structure to address a link prediction task, evaluating the likelihood of a model exhibiting high performance on the target dataset. The connectivity between the model and dataset vertices is established if the model is anticipated to yield favorable outcomes.

For effective resolution of the link prediction problem, it is imperative to distinguish positive edges from negative ones. In our pursuit of identifying high-performing models, we designate relationships where a model demonstrates good performance on the dataset as positive edges, while those with lower accuracy are categorized as negative edges.

We employ diverse graph learning algorithms for the acquisition of knowledge from the constructed graph. These algorithms consider a variety of information, e.g., link structure and edge attributes. In essence, graph learning algorithms demonstrate the capability to capture intrinsic knowledge within a graph by assimilating neighborhood information.

1) Random-walk-based graph learning algorithms: Graph learning algorithms based on random walks do not incorporate the features of vertices; instead, they focus on learning the graph's link structure. This paper specifically explores Node2Vec [30] and its variant, Node2Vec+ [31].

Node2Vec. Node2Vec [30] introduces a probability model where the random walk has a certain probability, $1/p$, to revisit nodes being traversed. Additionally, it employs an in-out parameter, q, to control the exploration of the global structure. When the return parameter, p, is small, the random walk may become trapped in a loop, focusing on the local structure. Conversely, when the in-out parameter, q, is small, the random walk resembles a depth-first-sampling strategy more closely, capable of preserving the global structure in the embedding space.

Node2Vec+. Node2Vec+ [31] is a variant of Node2Vec. Different from Node2Vec traversing the graph with parameters, p and q, Node2Vec+ takes into account the edge weights. When it constructs walks in the graph, the probability of visiting the next neighbor is associated with the edge weights.

2) Neural-network-based learning methods: Different graph neural networks can learn different kinds of information from the graph. All of them capture the edges in the graph. Some also learn from the edge attributes, or node features.

GraphSAGE. GraphSAGE [28] employs a sampling and aggregation method to perform inductive node embedding, utilizing node features such as text attributes, node profiles, and more. The model trains a set of aggregation functions that integrate features from the local neighbors and pass them to the target node, denoted as v_i. Subsequently, the hidden state of the node v_i is updated by:

$$h_i^{(k+1)} = ReLU\left(W^{(k)}h_i^{(k)}, \sum_{n\in N(i)}(ReLU(Q^{(k)}h_n^{(k)}))\right) \tag{4}$$

3) Attention graph embedding: We also consider another type of graph learning method, using attention mechanisms in the learning process. The attention mechanisms enable graph learning to concentrate on specific parts of a graph that are more relevant to a given task. One advantage of applying attention to graphs is the ability to filter out the

noisy components, thereby increasing the signal-to-noise ratio in information processing. In this line of work, we adopt graph attention networks (GAT) in this paper.

GAT. GAT [29] employs masked self-attentional layers to address the limitations of prior graph convolutional-based methods. The layers aim to compute attention coefficients.

$$\alpha_{ij} = \frac{exp(LeakyReLU(\overrightarrow{a}^T[W\overrightarrow{h_i}||W\overrightarrow{h_j}]))}{\sum_{k\in N_i} exp(LeakyReLU(\overrightarrow{a}^T[W\overrightarrow{h_i}||W\overrightarrow{h_j}]))}, \quad (5)$$

W is the weight matrix of the initial linear transformation. The transformed information for each neighbor's feature is then concatenated to derive the new hidden state. This new hidden state undergoes a *LeakyReLU* activation function, a widely utilized rectifier. The attention mechanism described above constitutes a single-layer feed-forward neural network, parameterized by the weight vector mentioned earlier.

VI. The Framework of TransferGraph

We propose *TransferGraph*, a framework that performs model selection via a graph learning process. There are a few steps in the graph-based model selection process, as shown in Figure 5. The processes are divided into four main steps:

A. Metadata and Feature collection

We first collect all the information needed, as described in Section IV. Step ①-④ indicate the collection process of different features and metadata used for the subsequent steps. Step ① obtains the dataset representations, which can be further applied to compute the similarity between datasets. Step ② encapsulates the training performance of models across different datasets, while step ③ represents the acquisition of transferability scores of models, which can be obtained from existing works, e.g., LogME [3]. Step ④ collects the metadata of models and datasets. All the collected information will be returned to the model zoo and stored as preparatory data for further processes.

B. Graph construction and learning

With the collected information, we continue to construct a graph in step ⑤, encapsulating relationships between models and datasets, and other attributes. The graph component and learning details are provided in Section V.

We embed different types of relationships in the graph. Datasets are connected to each other with edge weights encoding their similarity. Models are connected to datasets with weights of the training performance and/or transferbility scores. To preserve the graph's density and facilitate graph learning, we set specific heuristics during graph construction. These heuristics include setting thresholds to differentiate positive edges from negative ones, based on the edge weight. An edge between a model and dataset is established only when both the normalized fine-tune accuracy and the normalized transferability score meet or exceed the threshold. The heuristics and properties of the constructed graph are shown in Table II.

We further use one of the graph learners, e.g., Node2Vec, presented in Section V to capture the information in the graph, e.g., link structure or vertex features, as in step ⑥. The graph learner is trained for a link prediction task. With the trained graph learner, we extract the representations for each vertex, whose dimension is 128.

Table II: Statistics of the graph properties. (* indicates that the value vary when the dataset and model collection changes)

Graph property	Value 1	Value 2
Modality	image	text
graph type	homogenous	homogenous
Threshold on transferability score for edge pruning	0.5	0.5
Threshold on accuracy for edge pruning	0.5	0.5
Threshold of negative edge identification on accuracy	0.5	0.5
Number of nodes	265	188
Average node degree*	20.1	8.6
Number of dataset-dataset edge	5256	550
Number of model-dataset edge with accuracy weight*	1753	918
Number of model-datset edge with transferability weight*	916	419

C. Training prediction model to predict model performance

As a learning-based strategy, we learn from the training history to predict the model performance on an unseen dataset as a regression task. In step ⑦, we construct a training set for the supervised learning as a regression task. The label is the training performance of a model on a dataset. The training features are constructed by metadata of models and datasets, as well as the vertex representations of the models and datasets. For example, given the performance of model m_A on dataset d_B, we identify the metadata of m_A and d_B, as well as the vertex representations of them. The information is treated as features and train a prediction model.

The training set can be represented as tabular data. The prediction models are introduced below: We then can learn a prediction model, e.g., linear regression, random forest, on the prepared training set, as shown in step ⑧.

Linear regression. One of the prediction model we use is linear regression. We use the linear regression model to learn various features, e.g., meta features and graph features. Linear regression fits a straight line or surface that minimizes the discrepancies between predicted and actual output values.

Random forest. Random forest is also a highly adopted model due to its simplicity and explainability. We set the number of trees as 100, max depth as 5.

XGBoost. XGBoost (eXtreme Gradient Boosting) is one of the ensemble learning methods and is particularly effective in structured and tabular data scenarios [32]. XGBoost is an ensemble of decision trees and minimizes the objective function with gradient descent. We set the number of trees as 500, and maximum depth as 5.

D. Model recommendation for fine-tuning

We construct a prediction set ⑨ similarly to the training set construction. Specially, the dataset included in the prediction set is the target dataset we want to predict the model

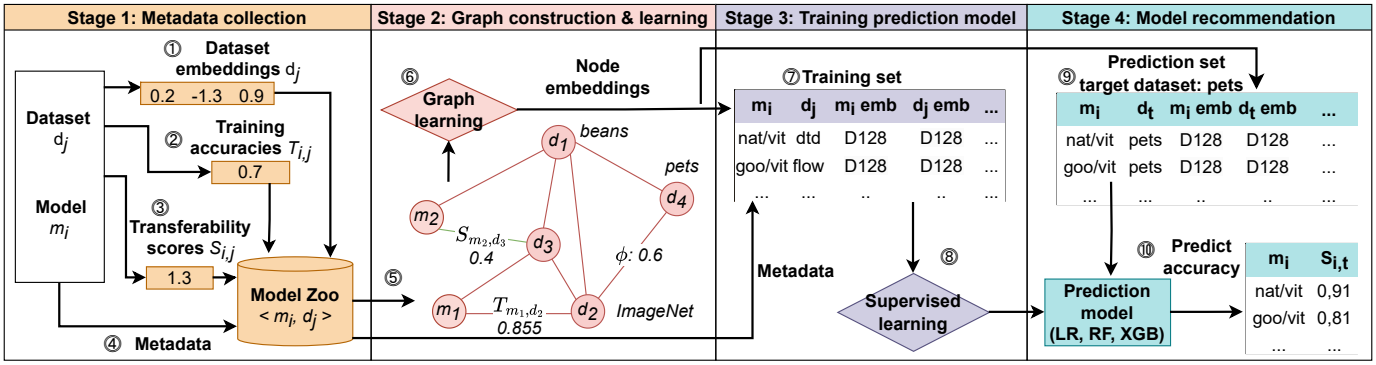


Figure 5: An overview of TransferGraph on model selection for fine-tuning, including model zoo construction (stage 1), training (stage 2-3) and model selection (stage 4).

performance on. We adopt a leave-one-out approach for the evaluation of our methodology. When training the prediction model, we utilize all the fine-tuning results from the pairs of models and datasets, excluding the target dataset. In the prediction set, we predict the performance of pairs between all models and the target dataset, i.e., d_t. The metadata of the dataset also adjust with the target dataset. We include all the models, since we would like to predict performance of the models in the model zoo on the target dataset. More details of the evaluation can be found in Section VII-A(Evaluation).

Given the trained prediction model, we obtain a score for each model and target dataset pair. We apply these predicted scores as an indicator to rank and select models for fine-tuning.

VII. Evaluation

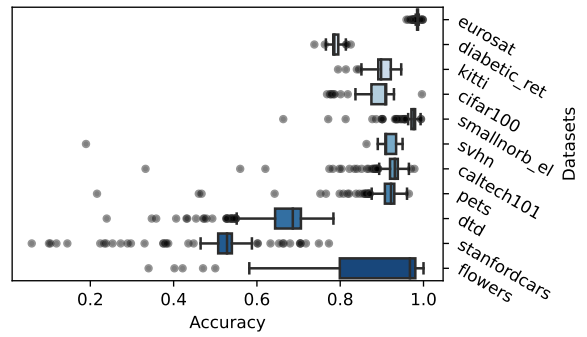
A. Experiment setups

Datasets. For evaluation, we collected 12 public image datasets and eight textual datasets, which are often used for classification benchmarks and are listed in Table III. We also included 61 image and 16 textual source datasets, which were used to compute dataset similarity.

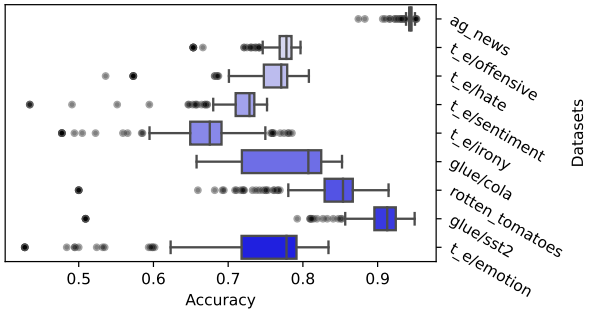
Models. We include 185 heterogeneous models for image classification tasks and 163 models for text classification tasks, with different architectures, such as ViT [22], Swin-Transformer [43] and ConvNeXT [44] for visual models and BERT [45], FNet [46] and ELECTRA [47] for NLP models, and pre-trained on diverse datasets. We use public image and text classification models from HuggingFace[5]. Different from the setup in the previous works [3], [9], we do not constrain the model to be trained only on a certain dataset, e.g., ImageNet.

Ground truth. A pre-trained deep learning model consists of two components: a *feature extractor* and a *classifier*. During fine-tuning, the model is initiated with the pre-trained weights, coupled with a classifier layer that is randomly initialized. Subsequently, this new model is retrained on the target dataset. To determine the actual fine-tuning accuracy, we fine-tune all models on our target datasets, using setups that generalize well over the different target datasets:

[5]https://huggingface.co/models



(a) Image datasets



(b) Textual datasets

Figure 6: Fine-tuning performance of models over different datasets sorted by standard deviation (*t_e* short for *tweet_eval*)

- For fine-tuning image classification models, we employ *stochastic gradient descent* in combination with a cyclical learning rate scheduler [48]. We optimize for 30 epochs, using a momentum of 0.9 and max learning rate of 1e-3.
- For text classification models, we used *AdamW* [49] in combination with a linear learning rate scheduler and optimized for five epochs, using betas (0.9, 0.999), epsilon 1e-8 and initial learning rate 5e-5.

We present the fine-tuning performance of models across different datasets, as in Figure 6. Notably, in certain datasets, the performance variance is small. For example, in the case of *eurosat*, where the standard deviation is only 0.005, model selection is not necessary. In the following experiments, we

Table III: The properties of the target datasets used for evaluation

Image dataset	caltech101 [33]	cifar100 [34]	dtd [35]	flowers [36]	pets [37]	smallnorb_elevation [38]	stanfordcars [39]	svhn [40]
Samples	3060	50000	1880	1020	3680	24300	8144	73257
Classes	101	100	47	10	37	18	196	10

Textual dataset	glue/cola [41]	glue/sst2 [41]	rotten_tomatoes	tweet_eval/emotion [42]	tweet_eval/hate [42]	tweet_eval/irony [42]	tweet_eval/offensive [42]	tweet_eval/sentiment [42]
Samples	8550	70000	10662	5050	13000	4600	24300	59900
Classes	2	2	2	4	2	2	18	3

only report results on datasets where model performance exhibit variation. The datasets are ordered by the standard deviation of the performance.

Baselines. We compare our work with the two baselines:

- `LR` (Amazon LR [9]) is the state-of-the-art approach for model selection for model zoos. It exploits the meta-features of models and datasets, and uses these features to train a linear regression model to predict the fine-tuned accuracy. The metadata of datasets includes data size, number of classes, etc. The metadata of models consists of the model architecture, input size, pre-trained domain, etc. `LR` indicates the strategy including only the metadata as features, `LR{all,LogME}` consisting metadata, dataset similarity and LogME score.

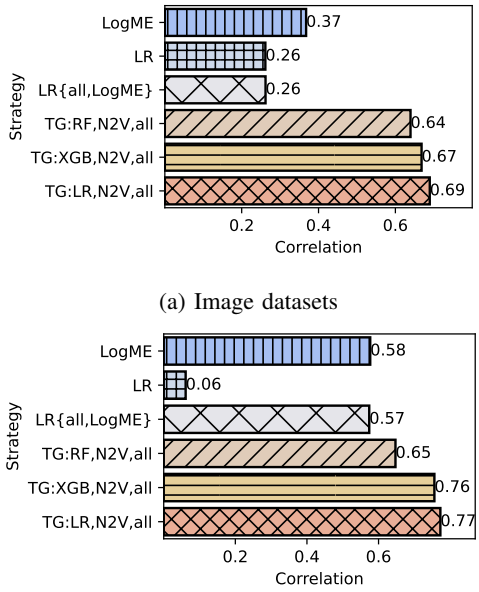
- `LogME` [3] is one of the most representative works that measure the transferability of a model to a target dataset. Transferability is a score that assesses a model's transfer learning performance to a new task (see Section VIII for explanations). The mechanism of LogME is to estimate the maximum value of label evidence $p(y|R)$ (R is the representations extracted by a model) given features extracted by pre-trained models.

Evaluation. To validate the effectiveness of our approach, we adopt a "leave-one-out" (LOO) mechanism for evaluation. This is a standard setting in related works of model selection, such as [9]. At each time, we learn from the training history of models trained on the existing datasets while excluding the target dataset. When constructing the graph in our proposed method, we remove all the edges of models connected to the target dataset node, i.e., the target dataset, while maintaining the edges between datasets. Then, with the learned GNN, we identify the node representations of models and the target dataset, and use them as the graph features.

For baseline comparison, we apply an evaluation metrics: *Pearson correlation*. Existing methods for model selection mostly predict a score, i.e., model selection score, for each pair of a model and target dataset. The Pearson correlation measures the correlation between the predicted scores and the ground-truth results, i.e., accuracy. A model selection method is considered better, with a higher correlation between its predicted score and the ground truth.

Summary of our proposed graph-learning-based strategy. There are a few design choices with our methods.

- **Prediction model** We include three prediction models: linear regression (`LR`), random forest (`RF`), XGBoost (`XGB`).
- **Graph learners**. The graph learning algorithms include GraphSAGE [28], GAT [29], Node2Vec [30] and Node2Vec+ [31]. In particular, `N2V(+)` is short for Node2Vec(+) in the plots.
- **Additional features for supervised learning**. Along with the graph features, we also include additional features as inputs for supervised learning when predicting the training results. We take into account features including all the metadata of models and datasets, as in Section IV-A2. In addition, we include the distance between the source dataset and target as another type of features for supervised learning, as in Section IV-B2.



(a) Image datasets

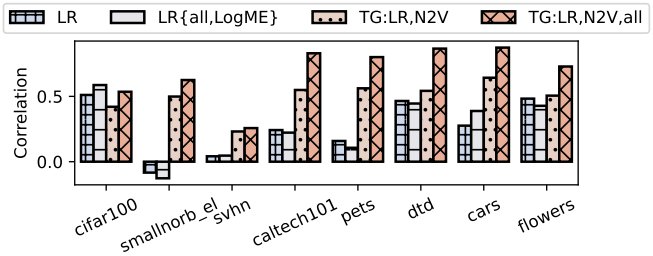
(b) Textual datasets

Figure 7: Comparing model selection strategies: feature-based (`LogME`), learning-based (`LR`), and our graph-learning-based (`TG`). Variants `LR` and `LR{all,LogME}` differ by feature use; `LR` applies basic metadata, whereas `LR{all,LogME}` includes metadata, dataset distance, and LogME scores. Our `TG` approaches use different predictive models with metadata, dataset distance, and graph features.

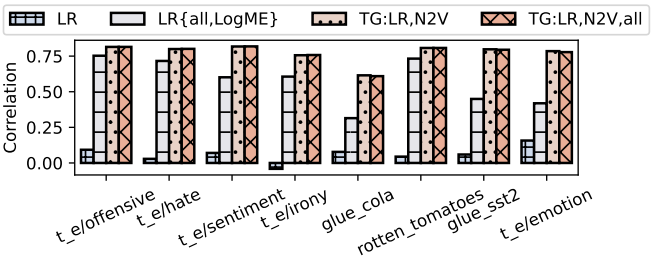
The variants of our proposed strategies begin with `TG`. For example, `TG:LR,N2V,all` indicates that we use a linear regression model *LR* to learn from *all* (including both the basic metadata and dataset similarity) supervised features along with the graph features obtained by Node2Vec *N2V*. `TG:LR,N2V` includes only the graph features obtained from Node2Vec.

B. Evaluation on heterogeneous model zoo

We first evaluate the effectiveness of our model selection strategies with the heterogeneous model zoo. In Figure 7, we



(a) Image datasets



(b) Textual datasets

Figure 8: Ablation study when including various features, i.e., i) with only metadata, ii) with metadata, dataset similarity and LogME scores, iii) with only graph-based features, and iv) with metadata, dataset similarity and graph-features.

report the average Pearson correlation between the predicted score and the fine-tuning results over 16 downstream datasets, eight for each modality, i.e., text and image. We compare our graph-learning-based strategy with other strategies mentioned in Section VII-A, i.e., `LogME`, and `LR`. `LogME` is feature-based and does not take into account of meta features nor the source dataset distance. The rest are all learning-based model selection strategies. They learn from the training history and predict the model performance on a target dataset. `LR` learns from the basic metadata, while `LR{all,LogME}` also includes the dataset similarity and transferability scores obtained by LogME. We present graph-featured-based strategies, beginning with `TG`.

Figure 7 shows that our proposed graph-feature-based strategies significantly improve the model selection performance compared to baselines `LogME`, `LR({all,LogME})`. We use three kinds of prediction models, i.e., linear regression model `LR`, random forest model `RF`, and XGBoost model `XGB`. Compared only using (meta) features (`LR`), the graph features can improve the capability of the prediction model and achieve a higher correlation between the predicted scores and the fine-tuning accuracy. It shows that adding additional information represented in a graph structure can help the prediction of model performance. We also notice that, on textual datasets, `LR{all,LogME}` significantly outperforms `LR`, which implies the significance of the selected features.

C. Ablation study

In this experiment, we conduct an ablation study where we investigate the effect of different features. We use the same prediction model, `LR` to learn from the features.

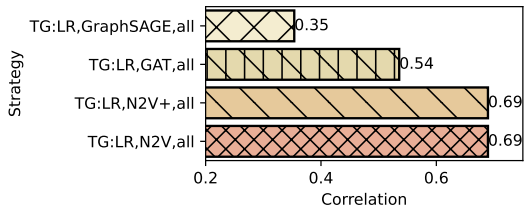


Figure 9: Performance of model selection strategies using different graph learners

As seen from Figure 8, including more features results in better performance. In particular,our approach using graph features significantly outperform the baselines. We note that when `LR` fails to learn, e.g., *smallnorb_evaluation*, the strategies using the graph features can successfully predict the model performance. Among all, the most effective strategy is to include all the features, i.e., metadata features, dataset distance, and graph features. We also notice that the adding graph features does not yield significant benefits on *cifar100*. We observe that the performance trends for models on *cifar100* exhibit variability, e.g., models that achieve medium performance on other datasets tend to underperform on *cifar100*. Future research could explore strategies to account for this performance variation, using it as prior knowledge to refine the approach.

Scenarios without training scores. It is worth noting that we explored scenarios lacking training history on image datasets, which is relevant in initial status. Here, we leverage transferability scores, like LogME, for performance estimation. Despite reduced information, our approach still outperforms baselines, achieving average correlations of 0.47 (with metadata, dataset similarity and graph features) and 0.42 (using only graph features).

D. Effect of graph learning methods

In the previous study, we investigate the effect of different features. We move forward to verify the effectiveness of different graph learning methods. In the following, we compare the average performance using different graph learning algorithms to extract the graph features. All the strategies included in this experiment learn an `LR` model to predict the fine-tuning performance. The graph features are extracted by four graph learners: i) *GraphSAGE* [28], ii) *GAT* [29], iii) *Node2Vec+* [31], and iv) *Node2Vec* [30].

Figure 9 presents the correlation results when using different graph features obtained by various graph learners. The strategies learning features from the Node2Vec series, i.e., *Node2Vec* and *Node2Vec+* outperform the ones using *GraphSAGE* and *GAT*. Each graph learners consume different graph properties. *Node2Vec* only learns the link structure. Besides the link structure, *Node2Vec+* also takes into account the edge attributes in the graph. While *GraphSAGE* and *GAT* obtain not only the link structure, edge attributes, but also the node features, each updating the node representations in different mechanisms.

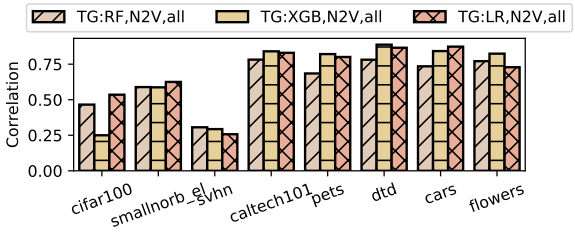


Figure 10: Effect of different prediction models

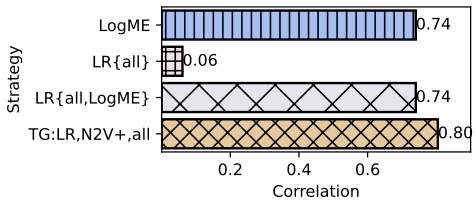
The graph neural networks usually work well on large graphs, e.g., *Citation data* containing 302,424 nodes, and *Reddit* with 232,965 edges [28]. *GraphSAGE* and *GAT* do not perform well in our context because the constructed graph is relatively small compared to those graph datasets. The graph used in this paper has only 265 nodes and thousands of edges. The computation overhead of obtaining such large-scale graph dataset is extremely expensive. While the *Node2Vec* series of graph learners can perform well on various size of graph dataset. It is noted that we do not explore the hyperparameter space of these graph learners, e.g., walk length, number of neighbors sampled by each node, window size, etc. Complementary work can identify the best hyperparameter candidate for each graph learners, and also investigate which graph learner to apply given different setting scenarios, e.g., graph size, link structure and node/edge features.

E. Effect and capability of prediction model

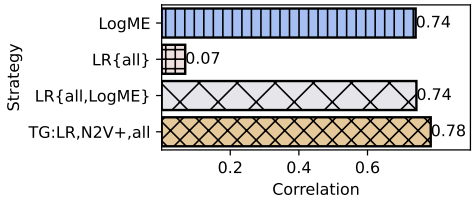
The prediction models are used to learn features and predict the fine-tuning scores. In this experiment, we investigate the effect when applying different prediction models. In Figure 10, we present the correlation performance when applying different prediction models. We observe that there is no dominant prediction model that can obtain the best results among all the datasets, and the performance of the prediction models on a dataset is similar in general, which indicates that the feature selection is more important than prediction model selection. Yet, we do not fully tune the prediction models on each dataset. Further study can be done to identify the most appropriate prediction model based on varying dataset characteristics.

F. Effect of fine-tuning method

There are multiple current practices to fine-tune models, each yielding different results. In the previous experiments, the fine-tuning method applied retrains all of the layers' pre-trained parameters, which is aiming at effectiveness, but expensive in terms of memory used. We adopt another fine-tuning method, *LoRA* [50], which is recently developed aiming at efficiency, both in time and memory. The mechanism is to freeze all model parameters and injects trainable rank decomposition matrices into each layer to reduce the number of trainable parameters. This enables the use of larger batch sizes and learning rates, achieves quicker convergence, but may lead to slightly reduced performance. We set an initial learning rate of 4e-4 and optimize for four epochs.



(a) New fine-tuning methods in both training and prediction stage



(b) Different fine-tuning methods in training and prediction stage

Figure 11: Comparison of model selection approaches with new fine-tuning method considered

To investigate the effect of different fine-tuning method, we repeat the experiments for the textual datasets using the new fine-tuning method. Below, in Figure 11, we show the results for both i) repeating the entire experiment with the new fine-tuning results, and ii) keeping the graph constructed with the previous fine-tuning results, but taking the new fine-tuning results as ground truth for the unseen dataset. We show that, for both settings, our graph-based approach consistently outperform the baselines. Compared to i), using different fine-tuning methods can result in slightly decrease in correlation performance, indicating that different fine-tuning methods do not impact the effectiveness of the approach a lot.

G. Discussion

Through comprehensive experiments, we have shown the efficacy of graph-based features in addressing the model selection problem with a model zoo. Our most competitive model selection strategy incorporates both graph-based features and additional metadata of models and datasets. It is noted that, in this paper, we use image classification task and visual models as illustrative scenarios. However, our proposed model selection strategy can be applied to diverse cases on various modalities. Below, we discuss the limitations and directions that can be investigated in future research.

Graph construction. We incorporate different information in a graph, e.g., dataset distance, model performance, dataset representations, etc. Yet, we do not discuss the contribution and importance of each type of features embedded in a graph. We apply a simple threshold-based edge pruning process to maintain the graph structure. Future work can investigate a more advanced graph construction method and make it adapt to the capability of different graph learning models.

Graph learning. We investigate four types of graph learner to obtain graph features. In the graph community, the performance of the graph learner may depend on the graph

properties. Further work can pursue to identify good candidates of graph learner (with tuned hyperparameters) for the graph generated from each specific model zoo. As future work, dynamic graph learning [51] can be investigated for continuous updates. The current approach requires retraining of the graph learner and regression model whenever new nodes or edges are added. By dynamically updating the graph learner, we extend TransferGraph to support timely update of the model recommendation. Moreover, future work can adapt methods [52], [53] to interpret and explain the graph learning process to improve the transparency of the model selection.

Integration with AutoML systems. Many modern AutoML tools or platforms focus on automatically finding the best hyperparameters or algorithms to train ML models [54], [55]. Our method could be used as a prior step, by first finding the best model to fine-tune. Organizations today already have access to platforms to train and host machine learning models[6,7]. We envision exciting future directions, where organizations build their private model zoos in combination with existing public pre-trained models. Through storing their (fine-tuning) metadata in a graph structure, integrating our proposed method could make recommendations for optimal future fine-tuning.

Complementary to RAG framework. Retrieval-Augmented Generation (RAG) [56] technique improves large language models (LLMs) by using an encoder model to index and retrieve documents from external sources. Our TransferGraph framework can be used to refine RAG components, offering fine-tuned models for both the encoder, which indexes documents, and the retriever, which fetches relevant document segments for queries. The fine-tuning ensures the encoder model is tailored to specific tasks [57], and the retriever delivers more relevant outcomes [58].

VIII. Related work

A. Transfer learning

Traditional machine learning techniques have seen significant progress in various knowledge engineering areas such as classification, regression, clustering and data mining. Despite these advancements, real-world applications frequently encounter limitations. Unfortunately, in many scenarios, obtaining sufficient and representative training data can be a costly and time-consuming effort. Transfer learning has been very successful in combatting these problems, especially in the domain of deep learning, where the *data dependence* is even greater [59].

The process of transfer learning typically begins with selecting an *upstream* or *pre-trained* model from a repository containing models trained on different source datasets and architectures. Subsequently, one or multiple selected models are then fine-tuned using the users' target dataset, and the user can select the fine-tuned model with the best characteristics for their downstream task. There are various available fine-tuning strategies identified by [59]. We adopt the most popular *network-based deep transfer learning* in this work. Network-based deep transfer learning refers to reusing the partial network that pre-trained in the source domain and retraining the deep neural network which used in target domain.

B. Graph learning

Graph learning broadly refers to machine learning on data structured as a graph. It is gaining more and more attention, as many complex real-world data can be expressed as graphs. Graph learning can be separated into four different methods: i) *graph signal processing*, ii) *matrix factorization*, iii) *random walk* and iv) *neural network* [60]. We focus on the latter two methods, as those are mainly used in graph learning-based recommender systems [17].

1) Random-walk-based graph learning algorithms: These types of algorithms sample random walks by traversing the graph. Given a walk length, i.e., number of steps, a random vertex is selected as the starting point and a neighbor vertex would be selected with probability as the next step in the walk. These walks indicate the context of connected vertices. The randomness of walks gives the ability to explore the graph and capture both the global and the local structural information by walking through neighboring vertices. After the walks are built, probability models, such as skip-gram [61], can be applied to learn the representations. The mechanism of the random-walk-based graph learning is aiming to make the representations of connected nodes in the vector space closer to each other while disconnected ones further away. In such a way, the representations capture the graph's intrinsic structure.

2) Neural-network-based learning methods: This line of works were inspired by the success of neural network models, RNNs and CNNs. Graph learning methods using RNNs resemble walks sampled from a graph as words, and use natural language processing models to learn representation of vectors. Another family of neural-network-based methods adopt CNN models. The input can be walks sampled from a graph or the entire graph itself. In this work, we only discuss CNN-based learning methods in this category. Representative works include GraphSAGE [28], GCN [62].

IX. Conclusion

We explore the use of a graph-learning-based model selection strategy within the model zoo framework and introduce a comprehensive framework to address the intricate model selection problem. Predicting model performance proves to be challenging, given no dominant model excels across all datasets. Extensive experiments have shown that effectiveness of leveraging the intrinsic relationships between models and datasets for predicting the model performance. The most competitive variant of our model selection strategy gains 32% increase in measuring the correlation of the predicted model performance and the fine-tuning results. Furthermore, the graph-learning-based model selection strategy can continuously be improved with more metadata and training history in the model zoo.

[6]https://huggingface.co/autotrain

[7]https://cloud.google.com/vertex-ai

REFERENCES

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, ISSN: 1063-6919.

[2] A. Deshpande, A. Achille, A. Ravichandran, H. Li, L. Zancato, C. Fowlkes, R. Bhotika, S. Soatto, and P. Perona, "A linearized framework and a new benchmark for model selection for fine-tuning," *arXiv preprint arXiv:2102.00084*, 2021.

[3] K. You, Y. Liu, J. Wang, and M. Long, "LogME: Practical assessment of pre-trained models for transfer learning," in *Proceedings of the 38th International Conference on Machine Learning*. PMLR, pp. 12 133–12 143, ISSN: 2640-3498.

[4] A. T. Tran, C. V. Nguyen, and T. Hassner, "Transferability and hardness of supervised classification tasks," pp. 1395–1405.

[5] C. Nguyen, T. Hassner, M. Seeger, and C. Archambeau, "LEEP: A new measure to evaluate transferability of learned representations," in *Proceedings of the 37th International Conference on Machine Learning*. PMLR, pp. 7294–7305, ISSN: 2640-3498.

[6] D. Bolya, R. Mittapalli, and J. Hoffman, "Scalable diverse model selection for accessible transfer learning," in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., pp. 19 301–19 312.

[7] L.-K. Huang, J. Huang, Y. Rong, Q. Yang, and Y. Wei, "Frustratingly easy transferability estimation," in *Proceedings of the 39th International Conference on Machine Learning*. PMLR, pp. 9201–9225, ISSN: 2640-3498.

[8] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" *Advances in neural information processing systems*, vol. 27, 2014.

[9] H. Li, C. Fowlkes, H. Yang, O. Dabeer, Z. Tu, and S. Soatto, "Guided recommendation for model fine-tuning," in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 3633–3642.

[10] F. Nargesian, E. Zhu, R. J. Miller, K. Q. Pu, and P. C. Arocena, "Data lake management: challenges and opportunities," *Proceedings of the VLDB Endowment*, vol. 12, no. 12, pp. 1986–1989, 2019.

[11] I. G. Terrizzano, P. M. Schwarz, M. Roth, and J. E. Colino, "Data wrangling: The challenging yourney from the wild to the lake." in *CIDR*. Asilomar, 2015.

[12] R. Hai, C. Koutras, C. Quix, and M. Jarke, "Data lakes: A survey of functions and systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 12, pp. 12 571–12 590, 2023.

[13] R. C. Fernandez, Z. Abedjan, F. Koko, G. Yuan, S. Madden, and M. Stonebraker, "Aurum: A Data Discovery System," in *ICDE*, 2018, pp. 1001–1012.

[14] Y. Zhang and Z. G. Ives, "Finding Related Tables in Data Lakes for Interactive Data Science," in *SIGMOD*, 2020, pp. 1951–1966.

[15] F. Nargesian, K. Q. Pu, E. Zhu, B. Ghadiri Bashardoost, and R. J. Miller, "Organizing Data Lakes for Navigation," in *SIGMOD*, 2020, pp. 1939–1950.

[16] C. Renggli, X. Yao, L. Kolar, L. Rimanic, A. Klimovic, and C. Zhang, "SHiFT: an efficient, flexible search engine for transfer learning," vol. 16, no. 2, pp. 304–316.

[17] Z. Wang, Z. Dai, B. Poczos, and J. Carbonell, "Characterizing and avoiding negative transfer," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 11 285–11 294.

[18] Y. Cui, Y. Song, C. Sun, A. Howard, and S. Belongie, "Large scale fine-grained categorization and domain-specific transfer learning," pp. 4109–4118.

[19] A. Achille, M. Lam, R. Tewari, A. Ravichandran, S. Maji, C. C. Fowlkes, S. Soatto, and P. Perona, "Task2vec: Task embedding for meta-learning," pp. 6430–6439.

[20] A. Deshpande, A. Achille, A. Ravichandran, H. Li, L. Zancato, C. C. Fowlkes, R. Bhotika, S. Soatto, and P. Perona, "A linearized framework and a new benchmark for model selection for fine-tuning."

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 2016, pp. 630–645.

[22] 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 *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

[23] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," vol. 86, no. 11, pp. 2278–2324, conference Name: Proceedings of the IEEE.

[24] A. R. Zamir, A. Sax, W. Shen, L. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, pp. 3712–3722.

[25] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[26] S. Black, G. Leo, P. Wang, C. Leahy, and S. Biderman, "GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow," Mar. 2021, If you use this software, please cite it using these metadata.

[27] J. Song, Y. Chen, X. Wang, C. Shen, and M. Song, "Deep model transferability from attribution maps," in *Advances in Neural Information Processing Systems*, vol. 32. Curran Associates, Inc.

[28] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, ser. NIPS'17. Curran Associates Inc., pp. 1025–1035.

[29] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks."

[30] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. Association for Computing Machinery, pp. 855–864.

[31] R. Liu, M. Hirn, and A. Krishnan, "Accurately modeling biased random walks on weighted networks using node2vec+," vol. 39, no. 1, p. btad047, publisher: Oxford University Press.

[32] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, 2016, pp. 785–794.

[33] 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 *2004 Conference on Computer Vision and Pattern Recognition Workshop*, pp. 178–178.

[34] A. Krizhevsky, "Learning multiple layers of features from tiny images."

[35] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, pp. 3606–3613.

[36] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*. IEEE, pp. 722–729.

[37] H. Zhang, S. Zhou, G. Y. Li, and N. Xiu, "0/1 deep neural networks via block coordinate descent," *CoRR*, vol. abs/2206.09379, 2022.

[38] Y. LeCun, Fu Jie Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 2. IEEE, pp. 97–104.

[39] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *2013 IEEE International Conference on Computer Vision Workshops*, pp. 554–561.

[40] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng, "Reading digits in natural images with unsupervised feature learning."

[41] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding," in *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, T. Linzen, G. Chrupa{\textbackslash}la, and A. Alishahi, Eds. Association for Computational Linguistics, pp. 353–355.

[42] F. Barbieri, J. Camacho-Collados, L. Espinosa Anke, and L. Neves, "TweetEval: Unified benchmark and comparative evaluation for tweet classification," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, T. Cohn, Y. He, and Y. Liu, Eds. Association for Computational Linguistics, pp. 1644–1650.

[43] 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 *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, pp. 9992–10 002.

[44] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A ConvNet for the 2020s," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, pp. 11 966–11 976.

[45] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, pp. 4171–4186.
[46] J. Lee-Thorp, J. Ainslie, I. Eckstein, and S. Ontanon, "FNet: Mixing tokens with fourier transforms," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, Eds. Association for Computational Linguistics, pp. 4296–4313.
[47] K. Clark, M.-T. Luong, and Q. V. Le, "ELECTRA: PRE-TRAINING TEXT ENCODERS AS DISCRIMINATORS RATHER THAN GENERATORS."
[48] L. N. Smith, "Cyclical learning rates for training neural networks," in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, pp. 464–472.
[49] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization."
[50] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-rank adaptation of large language models."
[51] J. You, T. Du, and J. Leskovec, "Roland: graph learning framework for dynamic graphs," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 2358–2366.
[52] S. Piaggesi, M. Khosla, A. Panisson, and A. Anand, "Dine: Dimensional interpretability of node embeddings," *arXiv preprint arXiv:2310.01162*, 2023.
[53] H. Yuan, H. Yu, S. Gui, and S. Ji, "Explainability in graph neural networks: A taxonomic survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 45, no. 5, pp. 5782–5799, 2022.
[54] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems 28 (2015)*, 2015, pp. 2962–2970.
[55] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19. Association for Computing Machinery, pp. 2623–2631.
[56] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023.
[57] P. Zhang, S. Xiao, Z. Liu, Z. Dou, and J.-Y. Nie, "Retrieve anything to augment large language models," *arXiv preprint arXiv:2310.07554*, 2023.
[58] X. V. Lin, X. Chen, M. Chen, W. Shi, M. Lomeli, R. James, P. Rodriguez, J. Kahn, G. Szilvasy, M. Lewis *et al.*, "Ra-dit: Retrieval-augmented dual instruction tuning," *arXiv preprint arXiv:2310.01352*, 2023.
[59] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *Artificial Neural Networks and Machine Learning – ICANN 2018*, ser. Lecture Notes in Computer Science, V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, and I. Maglogiannis, Eds. Springer International Publishing, pp. 270–279.
[60] F. Xia, K. Sun, S. Yu, A. Aziz, L. Wan, S. Pan, and H. Liu, "Graph learning: A survey," vol. 2, no. 2, pp. 109–127.
[61] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2013.
[62] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks."

APPENDIX

A. Effect of dataset representations

Task2Vec embeddings. Task2Vec [19] is another method that we implement to obtain node features. Unlike domain similarity, Task2Vec also takes into account the labels of the dataset and learns embeddings for different tasks with pre-trained networks. The main formula to retrieve the Task2Vec embeddings involves computing the diagonal Fisher Information Matrix of the network filter parameters for a given task:

$$\tilde{E}_k = F, \quad F = \mathbb{E}_{x,y\ \tilde{p}(x)p_w(y|x)[\nabla_w log p_w(y|x) \nabla_w log p_w(y|x)^T]}, \tag{6}$$

where F is the diagonal Fisher Information Matrix (FIM). The Task2Vec embedding can then be obtained by averaging the FIM for all weights in each filter of the probe network. This results in a fixed-size vector representation for each task that captures its complexity and semantic similarity to other tasks. The norm of this embedding correlates with the complexity of the task, while the distance between embeddings captures semantic similarities between tasks.

Experiments exploring the effect of dataset representations. We extract dataset representations using proposed methods from [19]. The representations are used in two ways: i) to compute the distance between datasets, and ii) used as initial node features for datasets. In this experiment, we aim to investigate the effect of different dataset representations. The dimension of a Task2vec embedding is 13842, and the one of a domain-similarity embedding is 1024, depending on the extraction layer of the reference model.

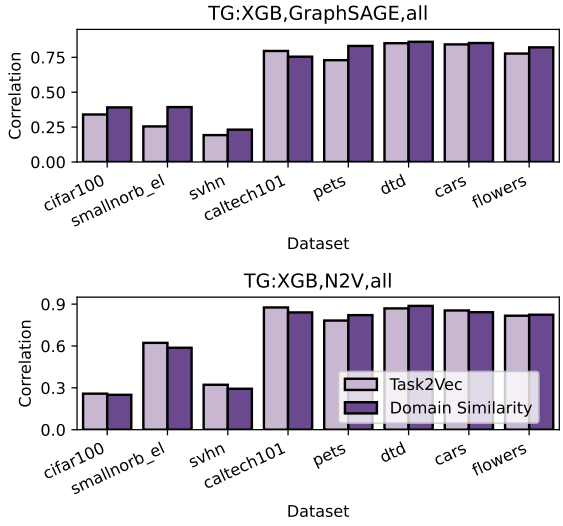


Figure 12: Correlation results affected by different dataset representations.

In Figure 12, we present the results of two of our proposed strategies, i.e., `TG:XGB,GraphSAGE,all` and `TG:XGB,N2V+,all`, using GraphSAGE and Node2Vec+ as graph learner respectively. We observe only slight differences

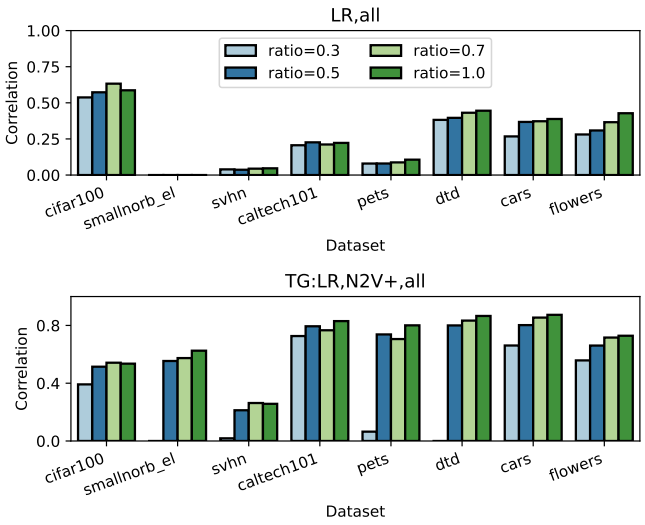


Figure 13: Effect of inputs ratio

in the performance on most of the datasets between using *Task2Vec* representations and the ones of *Domain Similarity*. For Nove2Vec+, the embeddings are only used to compute the dataset distance. The small differences in the dataset distance do not affect the final results. However, in the case of using GraphSAGE, where the representations are used for both similarity computation and the vertex features. We observe that in most cases, *Task2Vec* representations do not show advantages when using GraphSAGE. One reason is that the *Task2Vec* embeddings have really high dimension, while the graph in general is not big. We suggest that future work can delve into this and identify better representations for a graph learner to learn for the model selection problem.

B. Effect of input ratio

We investigate the effect of the input size of the training history on the performance. The entire training history ($ratio = 1.0$) include the training results of all the model and dataset pairs, excluding the target dataset and model pairs. We experiment on training with different ratios of the training history: $\{0.3, 0.5, 0.7, 1.0\}$. The strategy training would be much more efficient with lower input ratio, because the feature collection can be expensive though it can be performed offline. This experiment aims to answer the question: whether more training history help the prediction?

We compare two main categories, i.e., a strategy training without graph features (`LR,all`) and another strategy training with graph features (`TG:LR,N2V+,all`). As in Figure 13, the performance of both strategy can be affected by the input ratio. `LR,all` is more robust even when limited training history is used to train the strategy. While graph-feature-based strategy is more sensitive to the input ratio, especially with low input ratio. When we set training history as $ratio$=0.3, `TG:LR,N2V+,all` fails to predict the performance. The reason is that with a small input ratio, the constructed graph may have a large number of disconnected components. The graph learner fails to capture the global information by traversing the graph.