# CROSS-SELECT: A TRANSFORMER FRAMEWORK FOR PRE-TRAINED MODEL RECOMMENDATION

by

**Parth Vinod Patil**

**A Thesis**

*Submitted to the Faculty of Purdue University*

*In Partial Fulfillment of the Requirements for the degree of*

**Master of Science in Electrical and Computer Engineering**

School of Electrical and Computer Engineering

West Lafayette, Indiana

December 2025

# THE PURDUE UNIVERSITY GRADUATE SCHOOL STATEMENT OF COMMITTEE APPROVAL

**Dr. James C. Davis, Chair**

School of Electrical and Computer Engineering, Purdue University

**Dr. Qi Guo**

School of Electrical and Computer Engineering, Purdue University

**Dr. George K. Thiruvathukal**

Department of Computer Science, Loyola University Chicago

**Approved by:**

Dr. Milind Kulkarni

# ACKNOWLEDGMENTS

I wold like to thank Professor Davis for his wisdom and encouragement throughout my time working with him. His guidance has been invaluable, and I would not be the researcher I am today without it. I am especially thankful to my colleagues and friends from Purdue's Duality lab for their continuous support and advice. Finally, I am grateful to the support that has been given to me by my family, friends, and girlfriend.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# ABBREVIATIONS

| | |
|---|---|
| DNN | Deep Neural Network |
| NPM | Node Package Manager |
| PTM | Pre-Trained Model |
| PyPI | Python Package Index |

# ABSTRACT

The rapid expansion of pre-trained machine learning models (PTMs) has created substantial opportunities for reuse, while simultaneously introducing significant challenges for practitioners seeking to select models appropriate for their downstream tasks and deployment environments. As PTM repositories continue to grow in scale and heterogeneity, model selection has become increasingly complex, often requiring developers to consider interactions among task requirements, model properties, and hardware constraints. Existing PTM recommendation systems provide foundational progress but remain limited in two key respects: they typically overlook hardware-awareness, and current state-of-the-art approaches are constrained by fixed model zoos, limiting their ability to generalize effectively to newly emerging PTMs.

This thesis addresses these limitations through two complementary contributions. First, I propose a hardware-aware vision for PTM recommendation, articulating a concrete methodology, evaluation metrics, and system design principles that support more informed and context-sensitive model selection. Building upon this framework, the main contribution of the thesis advances the state of the art in PTM recommendation by introducing a cross-attentionbased transformer architecture designed to more effectively align task representations with model characteristics and generalize beyond predefined model repositories. Preliminary results indicate that this architecture achieves performance on par with, and in several cases exceeding, that of the current leading system (Model-Spider), demonstrating its improved adaptability and recommendation quality.

Overall, the findings highlight the importance of integrating hardware considerations into PTM selection workflows and demonstrate the viability of attention-based architectures for improving generalization in PTM recommendation. This work informs future research on PTM ecosystem analysis, automated model selection, hardware-aware reuse, and intelligent support for navigating the continually expanding landscape of pre-trained models.

# 1. INTRODUCTION

## 1.1 Overview

The rapid expansion of Pre-Trained Models (PTMs) has fundamentally reshaped modern machine learning workflows. Large public repositories now host millions of models across diverse modalities and architectures, enabling practitioners to build high-performing systems with reduced data and compute requirements. However, this abundance has introduced a significant engineering bottleneck: identifying which PTM is most suitable for a new dataset or downstream task. Model choice directly influences accuracy, efficiency, and deployability, yet current practice often relies on manual exploration or costly fine-tuning across many candidatesan approach that becomes increasingly impractical as the PTM ecosystem grows.

Existing automated recommendation approaches, such as ModelSpider, have begun to address this challenge by analyzing relationships between datasets and PTMs. Nevertheless, substantial limitations remain. Current methods generally ignore hardware and deployment constraints, despite their importance in real-world environments such as IoT devices with strict latency, memory, and energy budgets. Moreover, state-of-the-art recommenders are constrained by fixed model zoos and rely heavily on metadata or heuristic similarity measures, which restricts their ability to generalize to newly released PTMs and prevents them from capturing deeper semantic compatibility between datasets and models.

This thesis investigates automated PTM recommendation through two complementary contributions. First, it proposes a hardware-aware methodology that incorporates device-level constraints into the model selection process, improving its practicality for resource-constrained deployments. Second, it introduces a general learning-based framework that leverages transformer architectures with cross-attention mechanisms to directly model relationships between datasets and models and overcome the generalization limits of existing systems. Together, these contributions aim to make PTM selection more scalable, more accurate, and better aligned with the needs of real-world machine learning applications.

## 1.2 Motivation and Research Problem

Modern machine learning increasingly relies on reusing pre-trained models (PTMs), yet practitioners face three fundamental challenges when selecting an appropriate model for a downstream task.

First, the scale and diversity of todays model repositories make discovery itself difficult. With millions of PTMs differing in architecture, size, modality, training objectives, and data sources, practitioners lack systematic guidance for identifying which models are even plausible candidates. The consequence is an overwhelming and largely unstructured search space.

Second, evaluating candidate PTMs through fine-tuning is computationally prohibitive. Even tuning a single large model can require hours of GPU time, careful hyperparameter design, and repeated experimentation. Extending this process to tens or hundreds of modelsoften necessary for empirical selectionquickly becomes infeasible in practice. As a result, developers either overspend resources or settle for suboptimal models.

Third, existing PTM selection practices and automated recommenders are misaligned with deployment constraints. They prioritize accuracy-based metrics while largely ignoring the hardware budgets under which the model must operatememory capacity, compute throughput, latency requirements, and energy limitations. This omission is especially problematic in heterogeneous environments such as IoT deployments, where devices differ drastically in capability and many applications impose strict real-time or near-real-time constraints. Moreover, current recommenders often rely on fixed model zoos or shallow metadata and therefore cannot generalize to newly released PTMs or capture deeper datasetmodelhardware relationships.

These limitations highlight the absence of a unified, learning-based framework capable of predicting datasetmodelhardware compatibility before fine-tuning. This thesis investigates whether such a framework can be achieved through a data-driven approach that learns from large-scale PTM usage. The central research question is whether a deep learning model can reliably estimate PTM suitability in a way that reduces selection cost, improves generalizability, and enables more efficient model development across diverse deployment environments.

## 1.3 Thesis Statement and Contributions

**Thesis Statement:** This thesis proposes that effective and generalizable PTM recommendation can be achieved by (1) incorporating hardware- and deployment-aware considerations into the model selection workflow, and (2) employing cross-attentionbased transformer architectures that learn taskmodel relationships and generalize beyond fixed model zoos. Together, these components enable automated prediction of PTM suitability for diverse downstream tasks and emerging models.

The key contributions of this thesis are as follows:

- **A vision and empirical foundation for PTM recommendation on IoT devices.** This work, published in the SERP4IoT Workshop (2025), introduces the problem of model selection for IoT systems and proposes an early-stage framework for ranking PTMs according to device-level constraints such as latency, throughput, and energy.
- **A transformer-based PTM recommendation framework.** The second, ongoing work develops a data-driven pipeline that predicts the best PTM for a given dataset by learning joint representations of datasets and models. The system employs transformers with cross-attention to capture dataset-model interactions and leverages transfer learning to generalize across unseen tasks. Preliminary experiments show improved ranking accuracy over heuristic and metadata-based baselines such as ModelSpider.

**Significance of this thesis:** Pre-Trained Models represent a new paradigm in software and ML component reuse, but their rapid proliferation has outpaced our ability to manage and select them effectively. By proposing both a domain-specific framework for IoT and a general-purpose learning-based pipeline for PTM recommendation, this thesis contributes a foundational step toward intelligent PTM selection systems. It advances the vision of automating the selection stage of the PTM supply chain, complementing existing work that has largely focused on synthesis and deployment, thereby reducing the cost and complexity of integrating PTMs into real-world systems.

## 1.4 Outline of Remainder of Thesis

The remainder of this thesis proceeds as follows. In chapter 2, I present background on Pre-Trained Models, model reuse, and existing work in automated model selection, including metadata-driven and learning-based approaches such as ModelSpider. chapter 3 describes my first contribution: the vision and preliminary implementation of PTM recommendation for IoT devices, originally published in the SERP4IoT Workshop. chapter 4 presents my second contribution: a transformer-based PTM recommendation framework using cross-attention and dataset-model embedding alignment. Finally, chapter 5 discusses the broader implications of this research, its limitations, and future directions for building general-purpose PTM recommender systems.

## 1.5 Statement of Authorship, Attribution, and Copyright

This section clarifies ownership of material presented in this thesis. **??** includes material from my first-authored publication at the SERP4IoT Workshop (2025). **??** presents ongoing original work planned for submission to a peer-reviewed conference. I have adapted and expanded these materials to maintain consistency and context within the thesis. All figures and tables are my own unless otherwise noted.

# 2. BACKGROUND

This chapter provides a comprehensive review of the foundational knowledge that enabled the scaling of deep learning, the principles governing knowledge transfer between models and tasks, the metrics developed for efficient PTM selection, and the critical constraints and optimization strategies.

## 2.1 Deep Neural Networks and Transfer Learning

Deep neural networks (DNNs) have emerged as the dominant paradigm in machine learning, achieving state-of-the-art performance across diverse application domains including computer vision [1], [2], natural language processing [3], [4], speech recognition [5], and multimodal learning [6]. The success of DNNs stems from their capacity to learn hierarchical representations of complex, high-dimensional data through compositional layers of nonlinear transformations [7], [8].

### 2.1.1 Neural Network Fundamentals

A deep neural network consists of multiple layers of interconnected computational units (neurons), where each layer transforms its input through learned parameters. For a fully connected feedforward network, the transformation at layer $\ell$ can be expressed as:

$$\mathbf{h}^{(\ell)} = \sigma\left(\mathbf{W}^{(\ell)}\mathbf{h}^{(\ell-1)} + \mathbf{b}^{(\ell)}\right), \tag{2.1}$$

where $\mathbf{h}^{(\ell)} \in \mathbb{R}^{n_\ell}$ denotes the hidden state at layer $\ell$, $\mathbf{W}^{(\ell)} \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$ represents the weight matrix, $\mathbf{b}^{(\ell)} \in \mathbb{R}^{n_\ell}$ is the bias vector, and $\sigma(\cdot)$ is a nonlinear activation function such as ReLU [9], sigmoid, or GELU [10]. The initial layer receives the input data $\mathbf{h}^{(0)} = \mathbf{x}$, and the final layer produces the network output $\mathbf{y} = \mathbf{h}^{(L)}$ for a network with $L$ layers.

Modern DNN architectures extend this basic formulation with specialized structures tailored to specific data modalities and tasks. Convolutional Neural Networks (CNNs) [1], [11] employ spatially localized filters with weight sharing to process grid-structured data such as images, learning translation-invariant feature hierarchies. Recurrent Neural Networks (RNNs) [12] and their variants, including Long Short-Term Memory (LSTM) [13] and Gated Recurrent Units (GRU) [14], maintain hidden states across sequential inputs to model temporal dependencies in time-series and language data. The Transformer architecture [15] revolutionized sequence modeling by replacing recurrence with self-attention mechanisms, enabling parallel processing and capturing long-range dependencies more effectively. This architecture has become foundational for modern large language models [3], [4], [16] and vision transformers [17].

The parameters $\boldsymbol{\theta} = \{\mathbf{W}^{(\ell)}, \mathbf{b}^{(\ell)}\}_{\ell=1}^{L}$ are learned from training data through optimization algorithms that minimize a task-specific loss function $\mathcal{L}(\boldsymbol{\theta}; \mathcal{D})$ over a dataset $\mathcal{D}$. The predominant optimization approach is stochastic gradient descent (SGD) [18] and its adaptive variants such as Adam [19], RMSprop [20], and AdamW [21], which iteratively update parameters in the direction of the negative gradient:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}_t; \mathcal{B}_t), \tag{2.2}$$

where $\eta$ is the learning rate and $\mathcal{B}_t$ represents a mini-batch sampled from the training data at iteration $t$. Gradients are computed efficiently through backpropagation [12], which applies the chain rule to propagate error signals from the output layer back through the network.

Training deep neural networks from random initialization is computationally expensive and data-intensive, often requiring millions of labeled examples and substantial computational resources spanning days to weeks on specialized hardware such as GPUs or TPUs [1], [4]. The training cost scales with model size, dataset size, and architectural complexity, presenting significant barriers to entry for resource-constrained practitioners and application domains with limited labeled data [22], [23]. Furthermore, training from scratch provides no mechanism to leverage knowledge from related tasks or domains, necessitating redundant computation for each new application.

### 2.1.2 Transfer Learning

Transfer learning addresses the computational and data limitations of training DNNs from scratch by enabling models to leverage knowledge acquired from source tasks to improve performance on target tasks [24], [25], [26]. The fundamental premise is that representations learned on large-scale datasets for one task often capture generalizable features that remain useful across related tasks, particularly when the source and target domains share similar input distributions or underlying structures [27], [28].

Formally, transfer learning can be characterized by a source domain $\mathcal{D}_S$ with task $\mathcal{T}_S$ and a target domain $\mathcal{D}_T$ with task $\mathcal{T}_T$, where the objective is to improve the learning of the target predictive function $f_T(\cdot)$ using knowledge from $\mathcal{D}_S$ and $\mathcal{T}_S$ [24]. In the context of deep learning, this typically involves initializing a target network with parameters pre-trained on the source task, then adapting these parameters to the target task through one of several strategies [26], [28]:

- **Feature extraction**: The pre-trained network serves as a fixed feature extractor, where early layers remain frozen and only task-specific output layers are trained on target data [29], [30].
- **Fine-tuning**: All or a subset of pre-trained parameters are updated on the target task, allowing the network to adapt its representations while retaining useful source knowledge [27], [31].
- **Domain adaptation**: Specialized techniques explicitly align source and target distributions to minimize domain shift, often through adversarial training or distribution matching [32], [33].

The efficacy of transfer learning depends critically on the similarity between source and target tasks, both in terms of input distributions and task objectives [27], [34]. Recent work has developed quantitative measures to assess this transferability without exhaustive experimentation, including metrics based on feature distribution similarity [35], task relatedness scores [36], and spectral analysis of learned representations [34]. Zhang et al. [34] demonstrate that the distribution of spectral components in pre-trained representations provides a robust indicator of transfer learning potential, enabling more informed model selection without expensive empirical evaluation.

In natural language processing, transfer learning has fundamentally transformed the field through the paradigm of pre-training large language models on massive text corpora followed by task-specific fine-tuning [3], [4], [16], [28]. Models such as BERT [3], GPT [16], and their successors learn rich linguistic representations during pre-training that transfer effectively to downstream tasks including text classification, question answering, named entity recognition, and machine translation. Alyafeai et al. [28] provide a comprehensive survey of transfer learning techniques in NLP, highlighting the transition from task-specific architectures to general-purpose pre-trained models. Similarly, in computer vision, models pre-trained on ImageNet [37] have become standard initialization for a wide range of visual recognition tasks [38], [39].

The learnware paradigm [40], [41] extends transfer learning by treating pre-trained models as reusable knowledge units that can be identified and repurposed for new tasks without access to original training data. Guo et al. [41] address the challenge of identifying useful learnwares when source and target tasks have heterogeneous label spaces, enabling knowledge transfer even when task definitions differ substantially. This perspective aligns closely with the modern ecosystem of pre-trained model repositories, where thousands of models trained on diverse tasks are made publicly available for reuse [42].

The proliferation of pre-trained models across domains, architectures, and training regimes motivates the central problem addressed in this thesis: automatically recommending which pre-trained models are most suitable for a given target dataset or task, thereby reducing the manual effort and computational cost of model selection while maximizing transfer learning effectiveness.

## 2.2 Pre-Trained Models

Pre-trained models (PTMs) are neural networks trained on large-scale datasets whose learned parameters are made publicly available for reuse on downstream tasks, thereby amortizing the substantial computational cost of training across multiple applications [2], [3], [4]. This paradigm has become ubiquitous across domains: in natural language processing, models such as BERT [3], GPT [16], and T5 [43] pre-trained on massive text corpora enable state-of-the-art performance on diverse tasks through fine-tuning; in computer vision, architectures like ResNet [2] and Vision Transformers [17] pre-trained on ImageNet [37] provide robust feature extractors for image recognition; and in multimodal learning, models such as CLIP [6] bridge vision and language. The proliferation of PTM repositories including Hugging Face Hub [44], TensorFlow Hub [45], and PyTorch Hub [46] has democratized access to thousands of pre-trained models, significantly reducing barriers to entry for practitioners with limited computational resources or labeled data. However, this abundance introduces a critical challenge: determining which pre-trained model is most suitable for a given target task among the vast and growing collection of available options, motivating the need for automated model recommendation systems.

## 2.3 The Model Recommendation Problem

The democratization of pre-trained models has given rise to a paradox of choice: while repositories such as Hugging Face Hub now host over two million models [47], practitioners face the daunting challenge of identifying which model best suits their specific application without exhaustive empirical evaluation. This model recommendation problem manifests across three critical dimensions. First, the sheer scale of available models renders manual exploration infeasible, as even domain experts cannot meaningfully assess thousands of candidates varying in architecture, training regime, and task specialization [48]. Second, the computational cost of fine-tuning or even evaluating each candidate model on target data becomes prohibitive, particularly for resource-constrained practitioners who would benefit most from transfer learning [49]. Third, deployment constraints introduce additional complexity: hardware limitations, latency requirements, and energy budgetsespecially critical for edge devices and Internet-of-Things applicationsfurther restrict the feasible model space and necessitate recommendations that account for both task performance and operational constraints [50], [51].

The confluence of these challenges motivates the development of automated model recommendation systems that can efficiently predict model suitability without requiring exhaustive fine-tuning experiments [47], [52]. Such systems must balance predictive accuracy with computational efficiency, generalize across heterogeneous tasks and domains, and ideally incorporate deployment-aware considerations to ensure recommended models satisfy both performance and resource constraints. Addressing this problem requires moving beyond traditional model selection heuristics toward learning-based approaches that can capture complex relationships between model characteristics, dataset properties, and task requirements.

## 2.4 Approaches to PTM Recommendation

Researchers have developed various methods to help software engineers identify the most suitable PTMs, aiming to minimize fine-tuning and forward passes to reduce computational costs, Section 3.3.1 reviews state-of-the-art approaches for PTM recommendations in downstream tasks, categorizing them into two main types: heuristic- based and learning-based.

# 3. Recommending Pre-Trained Models for IoT Devices

## 3.1 Statement of Ownership

This chapter was conducted entirely by me as the first author alongside my co-authors. It has been published in SERP4IOT'25.

## 3.2 Introduction

Many IoT applications today use deep neural networks (DNNs) for advanced functionality. Examples include object detection in autonomous vehicles [53], crop health monitoring in agriculture [54], and natural language processing for smart home assistants [55]. Given the cost of developing and training a DNN from scratch [56], engineers often leverage pre-trained models (PTMs) to expedite development and deployment processes. However, as highlighted by recent studies [57], selecting an appropriate PTM frequently requires manual evaluation of model performance on downstream tasks, a process that is time-intensive and prone to variability. Furthermore, a separate study [58] found that practitioners often encounter hardware constraints or lack the expertise needed to adapt DNN as the main hurdle. This underscores the value of a systematic approach to PTM recommendation, particularly one that addresses the resource limitations of IoT hardware.

PTM selection extends beyond IoT, with prior work making significant strides in recommending PTMs for specific tasks without model fine-tuning. Figure 3.1 summarizes the current PTM selection pipeline. These methods fall into two categories: heuristic-based [59], [60], [61], [62], [63], [64], [65], [66] and learning-based [67], [68], [69], as detailed in 3.3. However, their primary focus is maximizing model performance (accuracy), with limited attention to hardware constraints. Deploying PTMs on IoT devices introduces many such constraints, including limited CPU, memory, energy, and low-bandwidth communication. Adapting current recommendation approaches to address these IoT-specific limitations is nontrivial, as many of these methods rely on measures such as class similarity between source and target tasks or the models capacity for distinguishing classes. Therefore, an effective IoT solution must incorporate both task suitability and hardware constraints, with

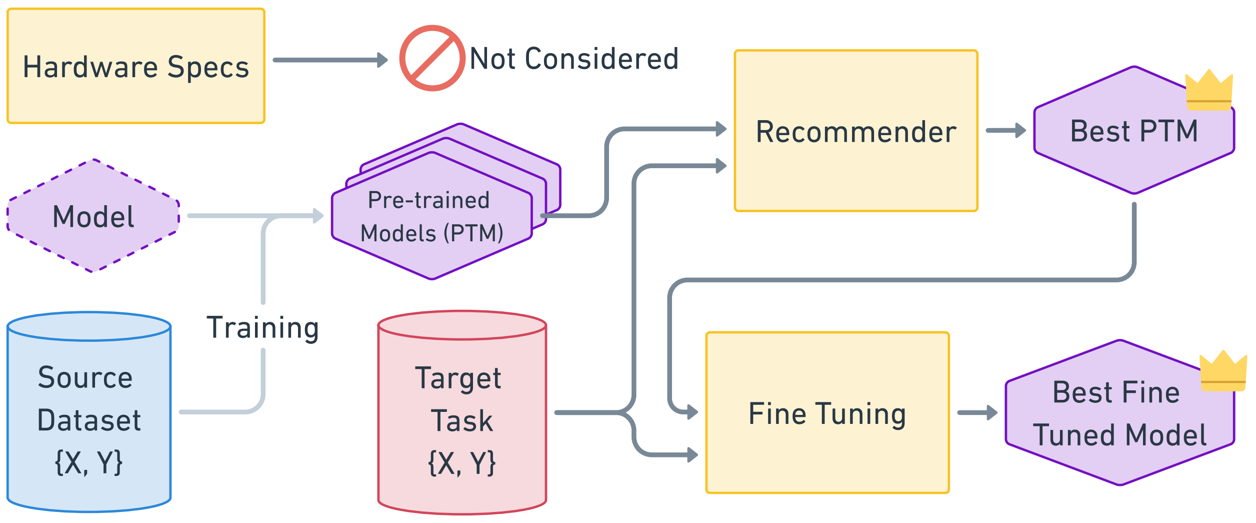


**Figure 3.1.** A model recommendation process for selecting the best pre-trained model (PTM) for a target task. Notably, hardware specifications are not considered, limiting use on constrained devices. Models trained on a source dataset (X, Y) are stored in a model hub. For a target task with a non-overlapping dataset (X, Y), the system recommends the most suitable PTM for fine-tuning, resulting in the best model for the task.

potential further insights from modeling device energy consumption. We identify two critical gaps in the state-of-the-art PTM recommendation systems: (1) *Lack of IoT-specific inputs for model recommendation* (2) *Lack of Ground-Truth rankings for IoT devices.* Developing a hardware-aware recommender requires first establishing a ground truth model ranking on devices, which is essential for addressing the first gap.

First, we propose two modifications to the Model Spider framework [67] to enable hardware-aware recommendations, which we term *Model Spider Fusion* and *Model Spider Shadow.* Both approaches aim to address the first gap, each with distinct methods. Additionally, we outline methods for collecting raw data by defining essential metrics and generating custom rankings based on these hardware-specific performance indicators. We close by discussing future opportunities to support the ongoing development of hardware-conscious model recommendation systems.

Our contributions are:

1. We identify key gaps in PTM recommendation for IoT.

2. We introduce methods to address these gaps, focusing on hardware-specific metrics and tweaks to existing methods.

3. We outline a research agenda aimed at advancing PTM recommendations with a focus on hardware and sustainability.

## 3.3 Background and related work

This section covers background and related works on pre-trained model recommendation (section 3.3.1) and model benchmarking in IoT systems (section 3.3.2).

### 3.3.1 Works for Model Recommendation

Researchers have developed various methods to help software engineers identify the most suitable PTMs, aiming to minimize fine-tuning and forward passes to reduce computational costs, as summarized in table 3.1. This section reviews state-of-the-art approaches for PTM recommendations in downstream tasks, categorizing them into two main types: *heuristic-based* and *learning-based*.

**Heuristic-Based Methods**: These methods typically devise a novel heuristic or scoring system to rank a PTM's effectiveness for a target task. This category includes methods, such as NCE [59], and H-Score [60], which estimate the similarity between source and target labels directly. Additional methods, such as LEEP [61], N-LEEP [62], LogME [63], PACTran [64], GBC [65], and LFC [66], rely on a forward pass on the target dataset to capture representations. These representations are then analyzed for their usefulness based on target labels. However, heuristic approaches often struggle to capture the nuances of hardware specifications and their correlation with model performance, overlooking the practical challenges software engineers face when deploying models on hardware-constrained environments. For example, on IoT devices, larger models can leverage hardware-accelerated activation functions to boost performance, while smaller models with complex functions may still encounter bottlenecks.

**Learning-Based Methods**: Recent methods, including Model Spider [67], EMMS [68], and Fennec [69], employ machine learning to eliminate the need for forward passes. These methods encode both the PTM and target dataset into a latent space, using learning techniques to discern patterns and predict performance. Given their promising results, we focus on learning-based techniques for our hardware-aware recommendation solutions, as this approach is well-suited for software engineers to encode complex hardware specifications, such as CPU type, architecture, memory size, and I/O speed.

Model Spider tokenizes tasks and pre-trained models to generate recommendations that balance efficiency and accuracy [67]. The Model Spider approach encodes each pre-trained model into a token $\theta_m$ using an extractor $\Psi$, which encapsulates essential characteristics such as architectures and parameters. Simultaneously, tasks are embedded as other tokens $\mu(T)$, reflecting dataset statistics and task characteristics. The key contribution of Model Spider is its use of a multi-head attention mechanism to assess the similarity $sim(\theta_m, \mu(T))$ between model and task tokens. This similarity score serves as a reliable indicator of model performance, enabling a ranked selection of pre-trained models aligned with the task requirements. Optionally, it also allows for the application of a forward pass, leading to a refined token $\theta_m^*$ that further captures data-specific attributes. This is done for top $K$-ranked models from the first iteration. Unlike EMMS [68] and Fennec [69], Model Spiders tokenized, attention-based design is more suitable to capture the nuances of hardware specifications. $^{B}emorespecific, whyit'smoresuitable.Becauseit'seasytoaddmorefeatures?$

**Table 3.1.** This table compares current model recommendation approaches and their requirements. ModelSpider is notably efficient, not requiring a forward pass on the target dataset; however, none address hardware constraintsa key limitation for resource-constrained IoT environments.

| **Recommendation Approach** | **Requires** | | | **Hardware Aware** |
|---|---|---|---|---|
| | *Forward Pass* | *Source Labels* | *Target Labels* | |
| NCE [59] | No | - | - | × |
| H-Score [60] | No | - | - | × |
| OTCE [70] | - | - | - | × |
| LEEP [61] | - | No | - | × |
| N-LEEP [62] | - | No | - | × |
| LogME [63] | - | - | No | × |
| PACTran [64] | - | - | - | × |
| GBC [65] | - | No | - | × |
| LFC [66] | - | No | - | × |
| | | | | |
| Model Spider [67] | No | No | No | × |
| EMMS [68] | No | No | - | × |
| Fennec [69] | No | No | - | × |
| | | | | |
| MS Fusion | No | No | No | Yes |
| MS Shadow | No | No | No | Yes |

### 3.3.2 Works for Model Benchmarking on IoT

Several studies have focused on benchmarking machine learning model performance on IoT devices, addressing specific applications and sustainability considerations relevant to software engineering workflows. For example, LwHBench [71] provides performance benchmarks to assess model performance, they use it for applications in agriculture, while Sayeedi

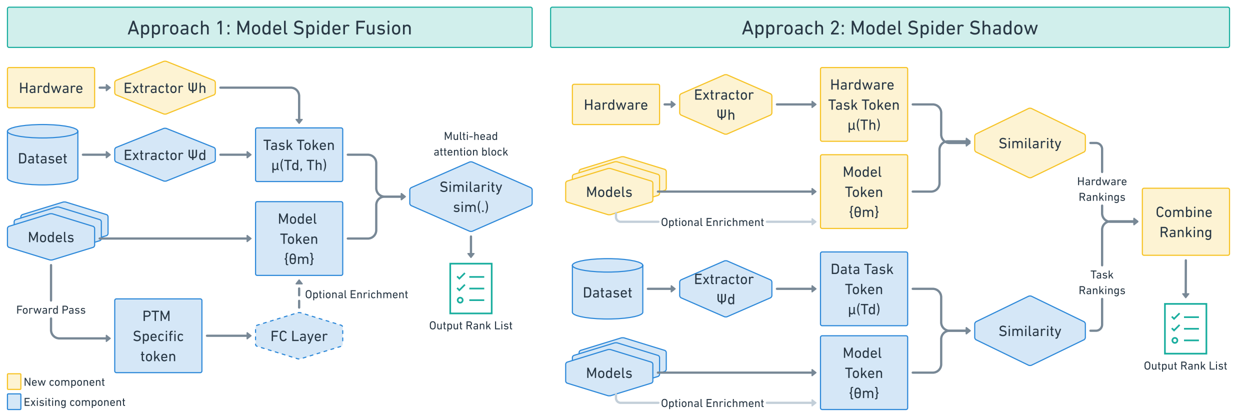


**Figure 3.2.** Overview of our proposed IoT-specific model recommendation approaches. Yellow components indicate new modules introduced, and blue represents existing Model Spider components. *Model Spider Fusion* incorporates hardware specifications directly into task tokens via a hardware extractor. *Model Spider Shadow* creates dual ranking systemstask relevance and hardware compatibilitycombined through Copelands method for balanced recommendations.

et al. [72] emphasize the sustainability of models in terms of environmental impact. However, these approaches primarily evaluate models individually rather than comparing and ranking them. Sayeedi et al. introduce innovative metrics such as the Green Carbon Footprint, which combines power consumption, execution time, and other factors to provide a more holistic assessment of a models environmental impact. Our work extends these studies by incorporating these metrics to provide a systematic way for software engineers to rank model performance across IoT devices, facilitating more informed and environmentally conscious model recommendations.

## 3.4 Motivation and Gap Analysis

While previous work has focused on model recommendations for downstream reuse, no studies or data currently support hardware-aware model selection. This section outlines key gaps in hardware awareness (section 3.4.1) and the lack of ground truth (section 3.4.2) in model recommender works.

### 3.4.1 Lack of IoT specific inputs for model recommendation

**A key limitation of current model recommendation approaches is that they fail to consider the hardware performance implications while selecting models.** For example, a recommendation system may rank models M1, M2, and M3, prioritizing M1 based on task performance. However, if M1 is a large model requiring extensive timesuch as 10 minutes for a forward pass on certain IoT hardwareit may be impractical for real-time applications, making M2 the more effective choice by balancing accuracy with practical performance constraints. Furthermore, deep learning compilers like ONNX could also influence model rankings by optimizing specific models for certain hardware types[73], [74]. Nevertheless, performance remains a major challenge for engineers when reusing PTMs, as highlighted by [75], and this issue is particularly critical in IoT contexts where resource limitations amplify its impact. As one participant in that study noted, "*Performance bugs are silent bugs. You will never know what happened until it goes to deployment.*" This emphasizes the need for a structured approach to assess a models overall performance on IoT devices.

To further explore the identified research gaps, we propose the following research questions:

### 3.4.2 Lack of Ground-Truth rankings for IoT devices

A fundamental barrier to progress in the earlier gap is the **lack of empirical data comparing the performance of different models across various IoT devices**. This gap prevents researchers from finding patterns or correlations in model parameters that could provide a quick, reliable way to assess hardware compatibility. Previous benchmarking works [71], [72], only consider one model and do not compare or rank them. The lack of this data restricts our ability to develop predictive models to recognize patterns in rankings. Model Spider [67] includes a method to approximate ground truth by combining multiple approaches using Copeland's rank aggregation [76], [77]. However, since there are few heuristic ranking approaches in the hardware space, this strategy is currently infeasible. This lack of comprehensive performance data across the hardware landscape highlights the need for dedicated datasets and benchmarks to enable informed and hardware-aware model recommendations.

The following research questions will guide our analysis after data collection, aiming to uncover trends and correlations:

## 3.5 Research Agenda

This section outlines a research agenda to enhance model recommendations for IoT applications. We introduce two approaches to make Model Spider hardware-aware (section 3.5.1) and propose methods for dataset creation (section 3.5.2) , improving adaptability across IoT environments. We then discuss extending our approach to broader deep learning systems and complex model reuse scenarios (section 3.5.3).

### 3.5.1 Modifications to Model Spider

To address the first gap identified, we propose two approaches to make the existing Model Spider framework hardware-aware. These build upon the existing framework, leveraging its robustness while enhancing its capability to account for hardware constraints with minimal modifications.

#### Model Spider Fusion – Augmenting Task Tokens

As shown in Figure 3.2, in this approach we propose to introduce a separate Extractor $\Psi_h$ which would encode the hardware specification for any given hardware. Its inputs comprise hardware model, CPU specifications, RAM size, Memory Size, etc. We would append these to the existing Task Tokens $\mu(T_d, T_h)$. This modification enables the similarity block to learn correlations between model performance and the specific hardware in use, thereby enhancing the hardware-awareness.

#### Model Spider Shadow – New Hardware task

This approach builds on Model Spider's capability to recommend the best model for a downstream task, expanding it to include hardware requirements. By redefining a "downstream task" to encompass specific hardware, we replicate the framework with a hardware

**Table 3.2.** Categorizing Metrics into Groups

| | Metric | Description | Sample |
|---|---|---|---|
| Hardware | Execution Time | Time for one forward pass | 90 ms |
| | Memory Utilization | Memory used during execution | 3 GB |
| | Power Consumption | Total energy consumed | 5 W |
| | CPU Temperature | Temperature of the CPU | 75řC |
| | Carbon Footprint [72] | Environmental impact | 10.9 units |
| Model | Accuracy | % correct predictions | 92% |
| | Precision | identified / predicted positives | 88% |
| | Recall (Sensitivity) | identified / actual positives | 85% |
| | F1 Score | Harmonic mean of last two | 86% |

extractor $\Psi_h$ that encodes hardware-specific features. This results in two ranking systems: a task-based selector for model suitability to tasks and a hardware-based selector for compatibility with hardware. We combine these rankings using Copelands method [76], [77], yielding a balanced recommendation that accounts for both task and hardware. This framework can be further extended to include energy-based or cost-based selectors, enabling tailored recommendations.

### 3.5.2 Dataset Creation

**Defining Metrics:** We propose to use metrics laid out in Sayeedi [72], categorized into two groups. These groups can either be combined to establish the ground truth in *Model Spider Fusion* (section 3.5.1) or used independently to rank models in *Model Spider Shadow* (section 3.5.1).

#### Methodology to collect data

To construct the ground truth data, we consider multiple PTMs and datasets. Each PTM is fine-tuned on all datasets. A methodology is then defined for collecting data from these (Model, Dataset) pairs, ensuring that experiments do not interfere with previous runs and that sensitive metrics are accurately reported.

#### Creating rankings from collected data

We propose a tunable ranking system that adjusts to specific requirements, ranking models based on selected metrics such as best execution time, best accuracy, or best energy consumption. To aggregate these metrics, we employ the weighted Copeland's rank-choice voting method. Each metric rank is assigned a weight $w_i$ in $[0, 1]$, with $\sum w_i = 1$. The objective function (1) maximizes each model's combined score, incorporating both model performance and hardware efficiency [78], [79]. Here, $f(\alpha)$ denotes model performance for configuration $\alpha$ (based on accuracy), and $\mathrm{HW}_i(\alpha)$ represents the $i$-th hardware metric (execution time or power consumption). Each hardware metric is normalized by a threshold $T_i$ and scaled by adjustable $w_i$ to reflect the relevant importance.

$$\max_{\alpha \in A} f(\alpha) \cdot \sum_i \left( \frac{\mathrm{HW}_i(\alpha)}{T_i} \right)^{w_i} \tag{1}$$

### 3.5.3 Future Directions

We propose to extend the model recommender to a broader deep learning system (section 3.5.3) and complex reuse (section 3.5.3).

#### Extending to Broader Deep Learning Systems

Our proposed approach would extend Model Spider, which handles only basic tasks (classification) on edge devices. However, hardware constraints are a crucial engineering consideration not only on edge devices but across a range of deep learning systems, including distributed learning frameworks (federated learning [80]). Additionally, in real-world applications, the interaction between data collection and model prediction components must be addressed [81], [82]. We propose broadening our research to include DL systems, enhancing model reuse and adaptability across diverse environments.

## Extending to More Complex Model Reuse Scenarios

Our work, along with most prior ML research, has primarily addressed basic tasks, such as classification and regression. Although Model Spider has explored model reuse in generative models (Large Language Model) [67], broader applications of DL model reuse remain largely unexplored. In computer vision, for instance, classification serves as a foundation for more complex tasks, like object detection and segmentation, which build upon classification models as backbones [83]. These downstream tasks require fine-tuning the model and adding task-specific heads or decoders for complex objectives. Expanding our approach to support such reuse scenarios is essential.

```
Input: Datasets and Models Fine-tuned on those
Output: Metrics contained in Table II
Function stabilize():
  Wait until:
    1) CPU and RAM utilization at nominal level
    2) Temperature within specified range
Function benchmark(Datasets, Models):
  Call stabilize()

  while dataset, model in pairs(Datasets, Models)
    for batch_size in range(1, 101)
      Sample batch with current batch_size
      Perform forward pass
      Measure performance metrics
    Set batch_size = 32
    for each batch in dataset
      Sample batch
      Perform forward pass
      Measure performance metrics
    Call stabilize()
```

## 3.6 Conclusion

In conclusion, this paper identifies and addresses critical gaps in PTM recommendation for IoT by proposing hardware-aware enhancements to the Model Spider framework. This work establishes a foundation for further research in optimizing PTM recommendations across diverse IoT environments.

# 4. Cross-Select: A Transformer Framework for Pre-Trained Model Recommendation

## 4.1 Statement of Ownership

This chapter was conducted entirely by me. It has not yet been published. It is in preparation for submission to peer review.

## 4.2 Introduction

Pre-trained models (PTMs) have become a central mechanism for reusing learned representations across tasks, particularly in computer vision where model zoos now contain dozens of architectures trained on diverse datasets. Choosing which PTM will perform best on a new target dataset, however, remains a non-trivial challenge. Direct evaluation of every candidate model is computationally expensive, and heuristic similarity measures—such as comparing label spaces or dataset statistics—often fail to capture the complex relationship between a datasets structure and a models learned capabilities. As a result, practitioners frequently rely on ad hoc intuition or costly trial-and-error procedures to identify a suitable model.

Recent learning-based approaches aim to automate this process by predicting transferability without requiring full fine-tuning. Model Spider, for example, learns a compatibility function between datasets and models and demonstrates that data-driven ranking can outperform classical heuristics. However, such methods require all candidate PTMs to be observed during training and rely on learned model embeddings that do not generalize to unseen architectures. This limitation poses a significant practical constraint, as modern model zoos evolve rapidly and continuously introduce new architectures, variants, and training regimes.

In this work, we introduce Cross-Select, a transformer-based framework that models dataset–model compatibility through a cross-attention mechanism. Unlike prior approaches, Cross-Select represents each model using an encoder applied directly to its weights and biases, allowing the system to generate model tokens for architectures never encountered during training. Similarly, dataset tokens are derived from representative samples without requiring task-specific fine-tuning. A cross-attention transformer then jointly processes dataset and model tokens, enabling the architecture to highlight fine-grained correspondences between dataset characteristics and model behavior. The resulting compatibility scores are used to rank models for the target dataset.

We evaluate Cross-Select across four generalization regimes that collectively test robustness to novel datasets, novel models, and fully out-of-distribution scenarios. Experiments on 11 structured datasets and 31 diverse vision models show that cross-attention substantially improves ranking accuracy, achieving strong correlation with ground-truth performance even when trained on limited supervision. Moreover, Cross-Select matches or exceeds the performance of Model Spider while using only a single image per class, demonstrating high data efficiency. However, our results also reveal a key limitation: although weight-derived tokens enable partial generalization, the framework struggles to fully handle unseen models, providing a clear direction for future work. Overall, Cross-Select provides a principled and extensible approach to PTM recommendation—one that moves beyond metadata, avoids restrictive assumptions about model availability during training, and leverages cross-attention to capture datasetmodel relationships in an end-to-end manner.

### 4.2.1 Background

#### PTM Repositories and Model Hubs

The growth of PTM hosting platforms has enabled wide accessibility but has simultaneously increased the complexity of model selection. The Hugging Face Hub contains a large and constantly changing collection of models, with model cards varying significantly in completeness and granularity. To support empirical study of PTM ecosystems, several datasets have been developed. HFCommunity provides structured snapshots of the Hugging Face Hub, including metadata, repository relationships, and usage statistics. PeaTMOSS further analyzes PTMs as software artifacts, examining distribution patterns, reuse practices, and dependency characteristics. PTMTorrent aggregates PTM packages across multiple repositories, enabling cross-hub analysis and highlighting fragmentation in PTM documentation and versioning. Collectively, these repositories illustrate that PTM selection occurs in a heterogeneous, large-scale environment where metadata cannot be assumed to be consistent or reliable.

### Transformer architecture

The Transformer architecture represents a paradigm shift in sequence modeling, replacing recurrent and convolutional layers with attention mechanisms as the fundamental building block [84]. At its core, the Transformer employs scaled dot-product attention, which computes attention weights by measuring the compatibility between queries $\mathbf{Q}$ and keys $\mathbf{K}$, then using these weights to aggregate values $\mathbf{V}$:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}, \tag{4.1}$$

where $d_k$ is the dimensionality of the key vectors, and the scaling factor $1/\sqrt{d_k}$ prevents the dot products from growing excessively large [85]. Multi-head attention extends this mechanism by projecting queries, keys, and values into multiple representation subspaces, allowing the model to jointly attend to information from different positions and representational aspects [86]. Each attention head learns distinct patterns, and their concatenated outputs are linearly transformed to produce the final representation.

Transformer architectures manifest in three principal variants, each suited to different tasks [87]. **Encoder-only models** apply self-attention bidirectionally across the input sequence, making them ideal for representation learning and classification tasks. **Decoder-only models** employ causal self-attention with autoregressive masking, restricting each position to attend only to previous positions, thereby enabling sequential generation tasks such as language modeling [88]. **Encoder-decoder models** combine both architectures, using self-attention within each component and introducing **cross-attention** in the decoder, where queries are derived from the decoder representations while keys and values originate from the encoder output [89]. This cross-attention mechanism enables the decoder to selectively attend to relevant portions of the encoded input, proving essential for tasks requiring alignment between two sequences, such as machine translation and sequence-to-sequence generation.

The cross-attention mechanism forms the foundation for our PTM recommendation framework presented in Section 4.3, where model embeddings serve as queries and dataset embeddings provide the keys and values, enabling the system to learn which model representations are most compatible with specific dataset characteristics.

### Model Spider Detailed overview

### Ranking and Evaluation Metrics

Evaluating the quality of predicted rankings requires metrics that quantify the concordance between two orderings. Kendall's rank correlation coefficient $\tau$ [90] measures the degree of agreement between two rankings by comparing the number of concordant pairs (pairs ranked in the same order) to discordant pairs (pairs ranked in opposite orders):

$$\tau = \frac{C - D}{\binom{n}{2}} = \frac{C - D}{n(n-1)/2}, \tag{4.2}$$

where $C$ denotes the number of concordant pairs, $D$ the number of discordant pairs, and $n$ the total number of items being ranked. The coefficient ranges from $-1$ (perfect disagreement) to $+1$ (perfect agreement), with 0 indicating no correlation [91].

For model recommendation tasks, not all ranking positions carry equal importance; accurately identifying the top-performing models is typically more critical than precisely ordering lower-ranked candidates. To address this, we employ weighted Kendall's tau [92], which assigns greater importance to pairs involving highly-ranked items:

$$\tau_w = \frac{\sum_{i<j} w_{ij} \cdot \operatorname{sgn}((r_i - r_j)(s_i - s_j))}{\sum_{i<j} w_{ij}}, \tag{4.3}$$

where $r_i$ and $s_i$ denote the ranks of item $i$ in the predicted and ground-truth rankings, respectively, and $w_{ij}$ represents the weight assigned to the pair $(i, j)$, typically defined as a decreasing function of rank positions [93]. This weighted variant serves as the primary evaluation metric for assessing the quality of PTM recommendations in our experimental results.

### Positioning of This Work

The proposed framework extends learning-based recommendation by removing the fixed-model constraint and replacing static similarity functions with transformer cross-attention between dataset and model embeddings. The system is designed to embed any candidate PTM using a general encoder and does not assume structured metadata or prior inclusion in the training pool. This enables general-purpose applicability across evolving PTM landscapes while preserving the representational benefits of learned compatibility scoring. The contribution therefore addresses limitations of both metadata-based and embedding-based recommenders and provides a scalable foundation for automated PTM selection.

## 4.3 Methodology

### 4.3.1 Design Overview

In Cross-Select, we introduce a transformer-driven framework that represents datasets and pre-trained models (PTMs) in respective latent spaces to estimate their compatibility without relying on exhaustive fine-tuning. The core idea is to embed both entities into fixed-size tokens, which can be passed to a transformer to deduce the similarity between the dataset and the PTM.

We begin by describing how Cross-Select obtains latent representations for datasets and PTMs. These embeddings serve as compact specifications that capture the salient statistical properties required for transfer learning. Building on these representations, we introduce a cross-attention module that models datasetmodel interactions: this mechanism allows the dataset embedding to selectively attend to PTM features that are most indicative of downstream performance.

Next, we calculate a compatibility score for each datasetmodel pair. This score reflects how well a given PTMs aligns with the structural and semantic characteristics of the target data set, enabling the system to effectively rank PTMs.

Finally, we discuss how the modular design of Cross-Select enables flexible extensions, such as incorporating model metadata, integrating PTMs beyond the original model zoo, or applying additional constraints. These capabilities enrich the PTM selection process and allow the framework to adapt to a broad range of use cases.

### 4.3.2 Obtaining Model and Dataset Tokens

This section elaborates on the token generation pipelines for PTM and datasets.

#### Generating Token for a Pre-Trained Model (ArchToVec)

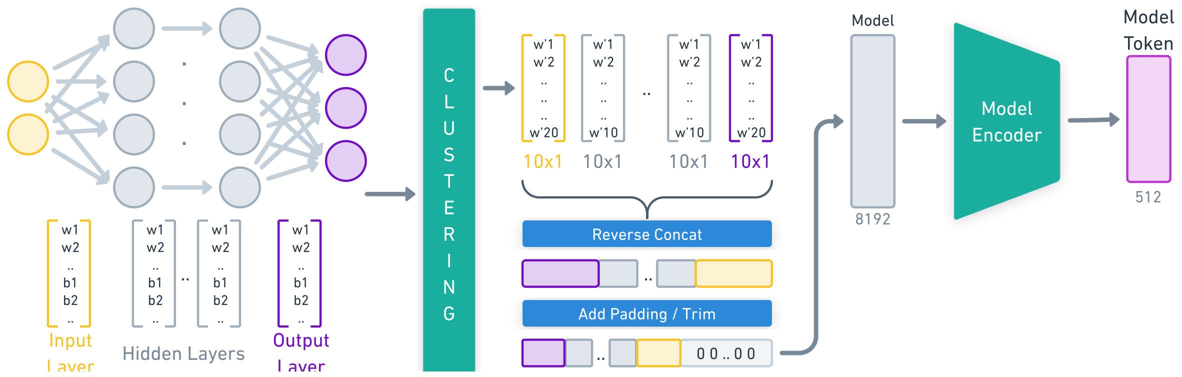


**Figure 4.1.** Model Compression and Encoding (ArchToVec) Pipeline. To standardize models with varying depths ($N_L$) and dimensions, layer weights are normalized via $k$-means clustering. The resulting centroids are concatenated in reverse layer order, padded to a fixed size, and compressed by an autoencoder ($AE$) into a compact model token $\theta$. This enables a unified semantic representation for diverse PTM architectures.

Consider a pre-trained model $M = \{W_\ell\}_{\ell=1}^{N_L}$, where each $W_\ell$ denotes the flattened weight–bias vector of the $\ell$-th layer. Since layers vary in shape both in dimensionality and in total number $N_L$ we first normalize their representations by performing layer-wise clustering. We use $k$-means to cluster the set of layer vectors. Formally,

$$\{C_\ell = \text{k-means}(W_\ell)\}_{\ell=1}^{N_L},$$

where $C_\ell$ denotes the $\ell$-th cluster centroid.

The resulting cluster centers are then ordered (in reverse layer order) and padded to match the fixed unencoded dimension. This standardized representation is then passed through an autoencoder $AE(.)$, which produces a compact model token of dimension $d$.

Formally, the final model token is obtained as

$$\theta = \mathrm{AE}\Big(\mathrm{Pad}\Big(\mathrm{Concat}\Big(\{C_\ell\}_{\ell=N_L}^{1}\Big)\Big)\Big).$$

This procedure is applied independently to every model in the model zoo, resulting in a collection of model tokens

$$\mathcal{M}_Z = \{\theta_i\}_{i=1}^{M},$$

where $M$ denotes the total number of available PTMs. And $M_z$ is a fixed Model Zoo

## Generating Tokens for a Dataset (Task Token)

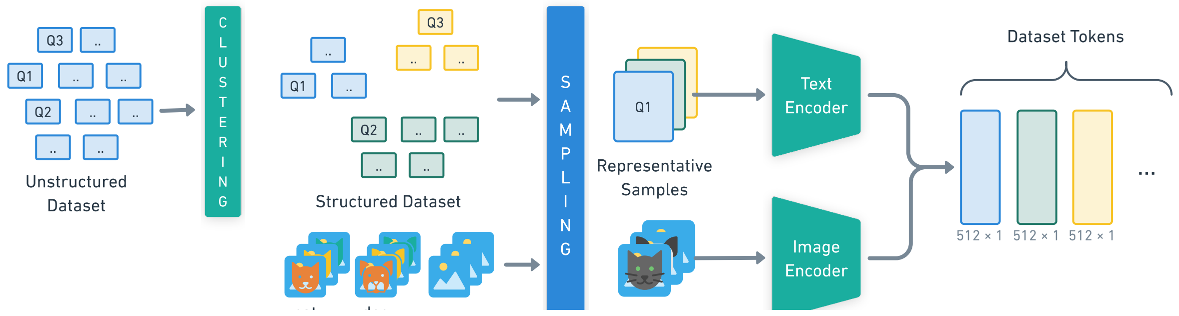


**Figure 4.2.** Dataset Tokenization Pipeline. To accommodate both labeled and unlabeled data, unlabeled samples are first clustered to generate pseudo-labels. Representative instances from each class are then processed by a frozen modality encoder (e.g., CLIP) to produce class-wise dataset tokens $\tau(D)$, which encapsulate the semantic structure and distributional characteristics of the target dataset.

Consider a dataset $D$, which may be labeled or unlabeled. For unlabeled datasets $D = \{x_i\}$, we first obtain pseudo-labels by clustering the samples using an unsupervised method. This produces a structured dataset $D = \{(x_i, y_i)\}$, where each cluster corresponds to a pseudo-class. Then for any structured dataset, from each class, we sample representative instances and encode them using a frozen modality encoder (e.g., a CLIP vision encoder). The resulting class-wise feature centers serve as the dataset tokens, capturing the semantic structure and distributional characteristics of $D$.

Formally, for each class $c$, the dataset tokens are computed as:

$$\exists i, \tau \quad \tau(D)_c = \psi(x_i)\, \mathbb{I}(y_i = c) \quad \forall \sphericalangle c \in C$$

where $\psi(\cdot)$ is the frozen encoder and $\mathbb{I}(\cdot)$ is the indicator function. The complete dataset token is the collection $\tau(D) = \{\tau(D)_c\}_{c \in \mathcal{C}}$. This approach naturally generalizes to datasets from other modalities. By replacing the encoder $\psi$ with an appropriate modality-specific encodersuch as a text encoder for language datasets, the same tokenization procedure can be applied without modification.

### 4.3.3 Similarity Estimation via Cross-Attention Transformer

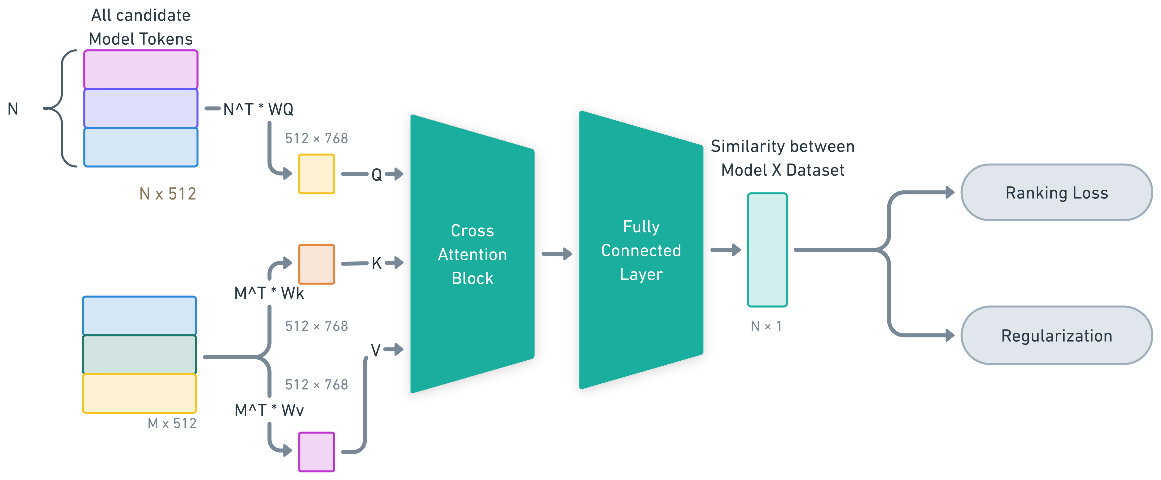


**Figure 4.3.** Transformer-Based Cross-Attention Architecture for PTM Recommendation. The architecture employs an Encoder-Decoder structure where dataset tokens are encoded to provide context (Keys/Values) and the model token drives the decoding process (Queries). This design allows the model representation to selectively attend to relevant dataset features through cross-attention layers, outputting a scalar similarity score.

To estimate the compatibility between a pre-trained model and a target dataset, we employ a Transformer-based architecture that takes advantage of cross-attention, denoted $\mathcal{T}_{cross}$. Unlike methods that rely on simple dot products or single-layer self-attention over concatenated tokens, our framework utilizes a deep Encoder-Decoder structure to explicitly model the directed relationship between dataset characteristics and model capabilities.

Consider a pair consisting of a sequence of dataset tokens $\tau(D)$ and a single model token $\theta_m$. The similarity computation proceeds as follows:

**Encoder (Dataset Context):** The dataset tokens $\tau(D)$ are fed into the Encoder stack. Each encoder layer applies multi-head self-attention followed by a feed-forward network, generating contextualized representations of the dataset's semantic structure.

**Decoder (Model Querying):** The model token $\theta_m$ serves as the input to the Decoder stack. In addition to a masked self-attention mechanism, each decoder layer incorporates a *cross-attention* block. In this block, the model token representation acts as the Query ($Q$), while the encoded dataset features serve as the Keys ($K$) and Values ($V$). This mechanism allows the model token to selectively "attend" to specific, relevant features within the dataset distribution.

**Architecture Depth and Output:** The Transformer $\mathcal{T}_{cross}$ consists of 6 stacked layers, enabling the learning of complex, non-linear interactions between the two modalities. The final output state of the decoder, which represents the model's alignment with the dataset, is passed through a Fully Connected (FC) projection head to produce a scalar similarity score:

$$\text{sim}(\theta_m, \tau(D)) = \text{FC}(\mathcal{T}_{cross}(\tau(D), \theta_m))$$

This score is subsequently used to rank all candidate PTMs for the target task.

**Zoo-Wide Scoring:** Since the Decoder processes the concatenated model sequence in parallel, the output is a corresponding sequence of feature vectors, one for each model. These are all passed through the same Fully Connected (FC) layer to produce a vector of similarity scores (or a probability distribution) for the entire zoo.

Formally, let $M_z = [\theta_1, \theta_2, \ldots, \theta_M]$ be the stacked tokens of all $M$ models in the zoo. The simultaneous scoring process is defined as:

$$\mathbf{S}_{zoo} = \mathrm{FC}\left(\mathcal{T}_{cross}^{\mathrm{Dec}}\left(M_z, \mathcal{T}_{cross}^{\mathrm{Enc}}(\tau(D))\right)\right)$$

where $\mathbf{S}_{zoo} \in \mathbb{R}^M$ is the resulting vector of compatibility scores (ranking PDF) for all candidate models given dataset $D$. This vector allows us to compute ranking losses and optimize the framework over the entire distribution of the model zoo in a single step.

### 4.3.4 Rank Prediction

To generate the final recommendations, the raw output scores for the entire model zoo, denoted as $S_{zoo}$, are first normalized. We apply a Sigmoid activation function to map the scalar outputs to a probability range $[0, 1]$, yielding the predicted compatibility scores $\hat{Y}$.

To determine the final ordering, these scores are sorted in descending order. We employ the *argsort* operation to extract the indices that correspond to the sorted probabilities:

$$\begin{aligned} \hat{Y} &= \sigma(S_{zoo}) \\ \hat{\pi} &= \mathrm{argsort}(\hat{Y})_{\mathrm{descending}} \end{aligned} \tag{4.4}$$

The resulting permutation $\hat{\pi}$ represents the predicted rank order of the candidate models in the zoo $\mathcal{M}_Z$, ordered by their suitability for the target task $\tau(D)$.

### 4.3.5 Objective Function: Listwise Ranking Loss

To optimize the parameters of the Cross-Select framework, we employ a listwise ranking loss function. This objective is designed to minimize the discrepancy between the predicted ranking order of the candidate models and the ground-truth performance ranking. Following the approach established in Model Spider, we assume that for a given training dataset $D$, we have access to a ground-truth ranking vector $\boldsymbol{\pi} \in \mathbb{R}^M$, where each element $\pi_m$ represents the ground-truth suitability score of the $m$-th pre-trained model.

Let $\hat{\boldsymbol{\pi}} \in \mathbb{R}^M$ denote the vector of predicted similarity scores for the $M_z$ candidate models, where the $m$-th component is given by $\hat{\pi}_m = \text{sim}(\theta_m, \tau(D))$. Our goal is to maximize the alignment between the predicted scores $\hat{\boldsymbol{\pi}}$ and the ground-truth scores $\boldsymbol{\pi}$.

To achieve this, we use a probabilistic ranking formulation. We define $\text{dsc}(\cdot)$ as an operator that sorts the indices of the ground-truth vector $\boldsymbol{\pi}$ in descending order. That is, if $\text{dsc}(m)$ returns the index of the model with the $m$-th highest ground-truth score, then $\pi_{\text{dsc}(1)} \geq \pi_{\text{dsc}(2)} \geq \cdots \geq \pi_{\text{dsc}(M)}$.

The ranking loss is formulated as the negative log-likelihood of the correct ranking order, calculated as follows:

$$\mathcal{L}_{rank}(\hat{\boldsymbol{\pi}}, \boldsymbol{\pi}) = \sum_{m=1}^{M} -\log\left(\frac{\exp(\hat{\pi}_{\text{dsc}(m)})}{\sum_{l=m}^{M}\exp(\hat{\pi}_{\text{dsc}(l)})}\right) \tag{4.5}$$

**Intuition:** This loss function enforces the correct relative ordering of the models. For each rank position $m$ (from best to worst according to the ground truth), the term inside the logarithm represents the probability that the model which *should* be at rank $m$ (i.e., model $\text{dsc}(m)$) is correctly identified as the best option among the remaining candidates (models $\text{dsc}(m)$ through $\text{dsc}(M)$).

By minimizing this loss, the model learns to assign higher similarity scores to PTMs that appear earlier in the ground-truth ranking, ensuring that the top-predicted recommendations correspond to the empirically best-performing models.

### 4.3.6 Training Procedure

The training of the Cross-Select framework is divided into two distinct phases. First, we train the **Model Autoencoder** to learn robust, compact representations (model tokens) from raw model parameters. Second, we train the **Cross-Attention Transformer** to rank these tokens based on their compatibility with dataset tokens.

## Phase 1: Training the Model Autoencoder

The objective of the Model Autoencoder is to compress high-dimensional layer parameters into a low-dimensional latent space while preserving structural information. The training data consists of flattened weight vectors extracted from the model zoo.

**Data Preprocessing:** Given a batch of raw weight vectors $\mathbf{w}$, we first apply standardization to ensure stable training dynamics. The inputs are normalized to zero mean and unit variance: $\mathbf{w}' = (\mathbf{w} - \mu)/(\sigma + \epsilon)$.

**Objective Function:** The network is trained to minimize a composite loss function consisting of a reconstruction term and a sparsity penalty. The reconstruction loss ensures the latent code captures essential features, while the L1 penalty on the latent code encourages sparse, interpretable representations.

$$\mathcal{L}_{AE} = \text{SmoothL1}(\mathbf{w}', \hat{\mathbf{w}}') + \lambda_{reg}\|\mathbf{z}\|_1$$

where $\hat{\mathbf{w}}'$ is the reconstructed input, $\mathbf{z}$ is the latent code (model token), and $\lambda_{reg}$ controls the regularization strength.

**Algorithm 1** Training the Model Autoencoder (ArchToVec)

```
Require: Set of extracted layer weights $\mathcal{W}$, Regularization weight $\lambda_{reg}$, Learning rate $\eta$
 1: Initialize Encoder $f_{enc}$ and Decoder $f_{dec}$
 2: while not converged do
 3:     Sample batch of weight vectors $\mathbf{w} \sim \mathcal{W}$
 4:     Normalize inputs: $\mathbf{w}' \leftarrow (\mathbf{w} - \mu)/(\sigma + \epsilon)$
 5:     Forward pass (Encoder): $\mathbf{z} \leftarrow f_{enc}(\mathbf{w}')$
 6:     Forward pass (Decoder): $\hat{\mathbf{w}}' \leftarrow f_{dec}(\mathbf{z})$
 7:     Compute Reconstruction Loss: $\mathcal{L}_{rec} \leftarrow \text{SmoothL1}(\hat{\mathbf{w}}', \mathbf{w}')$
 8:     Compute Sparsity Penalty: $\mathcal{L}_{reg} \leftarrow \lambda_{reg} \cdot \text{mean}($\mathbf{z}$)$
 9:     Total Loss: $\mathcal{L}_{total} \leftarrow \mathcal{L}_{rec} + \mathcal{L}_{reg}$
10:     Update parameters: $\theta_{AE} \leftarrow \theta_{AE} - \eta \nabla \mathcal{L}_{total}$
11: end while
12: return Trained Encoder $f_{enc}$ to generate Model Tokens $\theta_m$
```

## Phase 2: Training the Cross-Attention Transformer

Once the model tokens are generated, we freeze the autoencoder and train the Transformer to rank models. This phase utilizes the generated model tokens $\Theta_{zoo}$ and the dataset tokens $\tau(D)$.

**Temperature Scaling:** To improve ranking stability, we introduce a temperature parameter $T$ into the Softmax operation of the ranking loss. This temperature is annealed over the course of training (from an initial high value to a lower value), helping the model explore the ranking space early in training and sharpen its predictions in later epochs.

**Objective Function:** The Transformer is optimized using the listwise ranking loss describe by Equation 4.5, scaled by the current temperature $T$:

$$\mathcal{L}_{rank}(\hat{\boldsymbol{y}}, \boldsymbol{\pi}, T) = \sum_{m=1}^{M} -\log\left(\frac{\exp(\hat{y}_{\text{dsc}(m)}/T)}{\sum_{l=m}^{M} \exp(\hat{y}_{\text{dsc}(l)}/T)}\right)$$

**Algorithm 2** Training the Cross-Attention Transformer

```
Require: Dataset tokens $\{\tau(D_i)\}$, Model tokens $\Theta_{zoo}$, Ground truth ranks $\{\boldsymbol{t}_i\}$
Require: Temperature Schedule $T(step)$, Learning rate $\eta$
 1: Initialize Transformer $\mathcal{T}_{cross}$
 2: for epoch = 1 to $N_{epochs}$ do
 3:     for each batch $(\tau(D), m_{zoo}, \boldsymbol{\pi})$ in DataLoader do
 4:         Get current temperature: $T \leftarrow$ Scheduler($step$)
 5:         Forward Pass: For 6 layers
 6:         Encode Dataset: $K, V \leftarrow$ Encoder($\tau(D)$)
 7:         Decode Models: $\hat{\boldsymbol{s}}_{zoo} \leftarrow$ Decoder($m_{zoo}$, context = $(K, V)$)
 8:         Compute Loss:
 9:         Normalize Scores $\hat{y}_{zoo} = \sigma(\hat{s}_{zoo})$
10:         Calculate listwise ranking loss $\mathcal{L}_{rank}(\hat{\boldsymbol{y}}, \boldsymbol{\pi}, T)$
11:         Optimization:
12:         Update parameters: $\theta_{\mathcal{T}} \leftarrow \theta_{\mathcal{T}} - \eta \nabla \mathcal{L}_{rank}$
13:     end for
14: end for
```

## 4.4 Preliminary Experimental Setup

### 4.4.1 Overview

This section describes the preliminary experimental setup used to evaluate Cross-Select, our transformer-based framework for recommending pre-trained models (PTMs). Our study is designed to test two core hypotheses:

- (H1) Cross-attention architecture more effectively models the compatibility between dataset representations and PTM representations than self-attentionbased baselines
- (H2) Model tokens generated directly from weights and biases enable meaningful generalization to PTMs outside of model-zoo unseen during training.

To evaluate these hypotheses, we conduct experiments across four complementary generalization scenarios:

1. *ModelSpider Comparison* (known models and known datasets with unseen samples)
2. *Blind Dataset* (known models, new datasets)
3. *Blind Models* (new models, known datasets)
4. *Double Blind* (new models and new datasets)

Together, these quadrants assess the frameworks ability to generalize across both data distributions and model architectures. All experiments use standardized datasets and widely adopted PTMs drawn from open repositories such as HuggingFace and PyTorch Hub.

### 4.4.2 Training Data and Ground-Truth Rankings

We use 11 structured image datasets spanning three semantic domains: *Digits* (e.g., MNIST, USPS), *Fashion* (e.g., Fashion-MNIST, DeepFashion), and *Generic* (e.g., ImageNet-1k, Caltech101). In total, 31 pre-trained models form the model zoo, each trained on exactly one of the datasets. This setup allows the model to observe a broad range of datasetmodel relationships while maintaining clear provenance for each models training domain.

#### Selecting Datasets for Training

To ensure the recommender system learns to generalize across varying levels of task complexity, we selected 16 datasets in total. Of these, 11 were designated as **Training Datasets** (see Table 4.1). These were chosen to maximize diversity in scale and granularity:

- **Scale Variance:** The datasets range from small-scale tasks like *Caltech 101* (approx. 1,500 training images) to large-scale benchmarks like *ImageNet-1k* (1.2 million images).
- **Class Granularity:** The number of classes varies from 10 (e.g., *CIFAR10*, *MNIST*) to 1,000 (*ImageNet*), ensuring the system learns to handle both coarse and fine-grained classification tasks.

**Note on Testing Data:** The remaining 5 datasets (*Aircraft, Cars, Pets, DTD, SUN397*) are strictly reserved as **Blind Datasets**. These are never seen during training and are used exclusively to evaluate the "Blind Dataset" generalization scenario.

**Table 4.1.** Dataset Selection Overview. Datasets are categorized by genre and designated for either Training (Known) or Testing (Blind) phases.

| **Name** | **# Classes** | **Genre** | **Used For** |
|---|---|---|---|
| Cifar10 | 10 | Generic | Training |
| Cifar100 | 100 | Generic | Training |
| Caltech 101 | 101 | Generic | Training |
| Imagenet 1k | 1,000 | Generic | Training |
| FashionMNIST | 10 | Fashion | Training |
| FashionProduct | 30 | Fashion | Training |
| Deepfashion | 25 | Fashion | Training |
| Ham10000 | 7 | Medical | Training |
| SVHN | 10 | Digits | Training |
| MNIST | 10 | Digits | Training |
| USPS Digits | 10 | Digits | Training |
| Aircraft | 100 | Fine-Grained | Testing |
| Cars | 196 | Fine-Grained | Testing |
| Pets | 37 | Fine-Grained | Testing |
| DTD | 47 | Texture | Testing |
| SUN397 | 397 | Scene | Testing |

### Selecting PTM for the Model Zoo

The Model Zoo was constructed to represent a realistic cross-section of convolutional architectures, ranging from lightweight mobile-friendly models to deep, parameter-heavy networks. We selected 41 models in total, split into a training zoo and a testing set (Table 4.2).

**Diversity and Selection Logic:**

- **Architectural Variety:** The zoo includes distinct families such as *ResNet* (varying from 20 to 1202 layers), *VGG* (with and without batch normalization), *DenseNet*, and *MobileNet*.

- **Parameter Spread:** Model sizes range from as few as 0.01M parameters (*resnet20*) to over 86M parameters (*googlenet*), necessitating that the recommender learns to map task difficulty to model capacity.

**Note on Testing Models:** The 31 models marked as "Selected" form the fixed Model Zoo used during training. The remaining 10 models (including *ShuffleNet*, *EfficientNet*, and *RegNet*) are reserved as **Blind Models** to evaluate the system's ability to generalize to unseen architectures.

### Ground-Truth Ranking Construction

For each target dataset $d_t$, we compute a ground-truth ranking over models using a score function $s_k$ for the $k$-th model $m_k$. This function accounts for both the model's native performance and the domain proximity between the model's source dataset and the target task:

$$s_k = \begin{cases} a_{d_t,m_k} + 1, & \text{if } m_k \text{ is trained on } d_t, \\ a_{d_b,m_k} \cdot \sigma(d_t, d_b), & \text{if } m_k \text{ is trained on another dataset } d_b \neq d_t. \end{cases}$$

Here, $a_{d,m}$ denotes the models reported accuracy on its own training dataset, and $\sigma(d_t, d_b)$ is a pre-computed similarity score between datasets. These similarity scores $\sigma$ reflect structural and semantic alignments across domains (e.g., *MNIST* to *USPS* has high $\sigma$, whereas *MNIST* to *ImageNet* has low $\sigma$).

This produces a scalar compatibility score for each model, which is then converted into a dense ranking:

$$\pi = \text{argsort}(s + \varepsilon)_{\text{desc}},$$

where $\varepsilon$ is a small random perturbation used only to break ties.

### 4.4.3 Evaluation Metrics

To assess the quality of recommendations, we employ ordinal correlation metrics, specifically the weighted variation of Kendall's Rank Correlation $\tau_w$.

**Table 4.2.** Complete Model Zoo Composition. The table lists all 41 models split across two columns for balance. Models primarily used for **Training** are listed first, followed by models reserved for **Testing**.

| **Name** | **Arch.** | **Trained On** | **Used For** | **Name** | **Arch.** | **Trained On** | **Used For** |
|---|---|---|---|---|---|---|---|
| vit-base-mnist | ViT | MNIST | Training | vit_large_patch16_224...augreg | ViT-Large | ImageNet 21k | Training |
| clip-vit-base-patch32_mnist | CLIP | MNIST | Training | resnet50.deepfashion | ResNet50 | DeepFashion | Training |
| Mnist-Digits-SigLIP2 | SigLIP | MNIST | Training | resnet50.fashion_mnist | ResNet50 | FashionMNIST | Training |
| clip-vit-base-patch32_svhn | CLIP | SVHN | Training | resnet50.fashion_product | ResNet50 | FashionProduct | Training |
| resnet34_svhn | ResNet34 | SVHN | Training | resnet50.ham10000 | ResNet50 | Ham10000 | Training |
| densenet121_svhn | DenseNet121 | SVHN | Training | resnet50.svhn | ResNet50 | SVHN | Training |
| vgg16_bn_svhn | VGG16_BN | SVHN | Training | resnet50.mnist | ResNet50 | MNIST | Training |
| Fashion-Mnist-SigLIP2 | SigLIP | FashionMNIST | Training | resnet50.usps_digit_classifier | ResNet50 | USPS Digits | Training |
| fashion_classification_model | ViT | FashionMNIST | Training | shuffle_v2_x1.0 | ShuffleNet | ImageNet 1k | Testing |
| clip-vit-base-patch16_fashion_mnist | CLIP | FashionMNIST | Training | shuffle_v2_x1.5 | ShuffleNet | ImageNet 1k | Testing |
| vit-base-patch16-224 | ViT | FashionMNIST | Training | shuffle_v2_x2.0 | ShuffleNet | ImageNet 1k | Testing |
| eva02_large_patch14_448...m38m | EVA02 | ImageNet | Training | effnet_b0 | EfficientNet | ImageNet 1k | Testing |
| eva02_large_patch14_448...in22k | EVA02 | ImageNet | Training | effnet_b1 | EfficientNet | ImageNet 1k | Testing |
| eva_giant_patch14_560...m30m | EVA | ImageNet | Training | effnet_b2 | EfficientNet | ImageNet 1k | Testing |
| eva02_large_patch14_448...ft_in1k | EVA02 | ImageNet | Training | effnet_b3 | EfficientNet | ImageNet 1k | Testing |
| eva02_base_patch14_336...ft_in1k | EVA02 | ImageNet | Training | regnet_y_400m | RegNet | ImageNet 1k | Testing |
| eva02_large_patch14_336...m38m | EVA02 | ImageNet | Training | regnet_y_800m | RegNet | ImageNet 1k | Testing |
| vit_base_patch16_384.mim_in1k | ViT | ImageNet 1k | Training | regnet_y_1.6g | RegNet | ImageNet 1k | Testing |

**Kendall's Rank Correlation ($\tau$).** Kendall's Tau quantifies how closely the predicted ranking aligns with the ground-truth ordering derived from empirical performance measurements. The statistic counts the proportion of *concordant* pairs (+1) versus *discordant* pairs (–1), yielding a normalized score in the range $[-1, 1]$. Higher values indicate stronger agreement with the true performance order.

**Weighted Kendall's Tau ($\tau_w$).** We report the weighted variant of Kendalls Tau, which applies a hyperbolic weighting function that emphasizes correctness at the top of the list. Discrepancies among the highest-ranked items (e.g., ranks 0–2) contribute significantly to the score, while disagreements deeper in the list have negligible impact. This weighting reflects practical deployment settings in which only the top few recommendations matter most.

### 4.4.4 Hardware and Environment

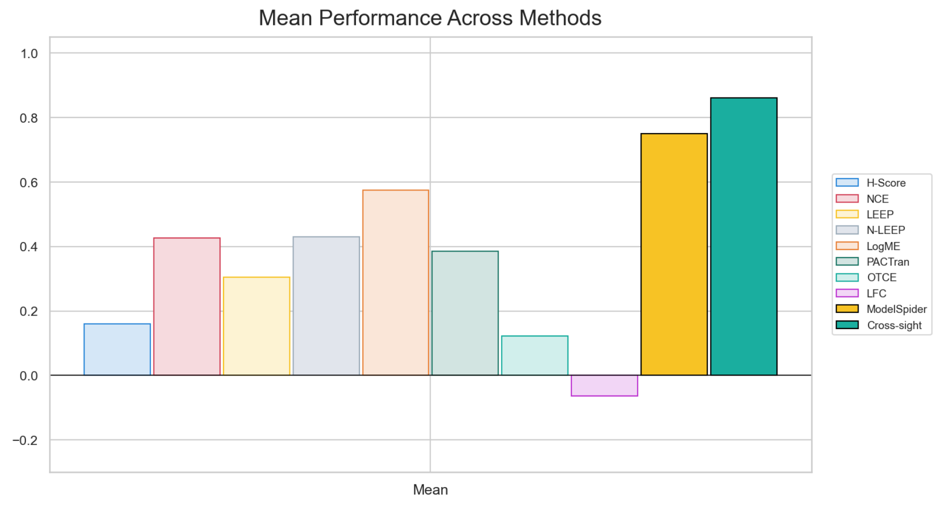
All Cross-Select training and evaluation experiments were conducted on Purdue Universitys Gautschi computing cluster. The training environment consists of a single NVIDIA L40S GPU with 40 GB of memory and a host machine equipped with 64-core CPU. Model training required approximately 10 hours, with each epoch comprising 37,904 optimization steps and a total training schedule of approximately 300–400 epochs.

The implementation is based on PyTorch and CUDA, with standard tooling for experiment management and reproducibility. All computations, data preprocessing, and model checkpoints were executed within this controlled cluster environment.

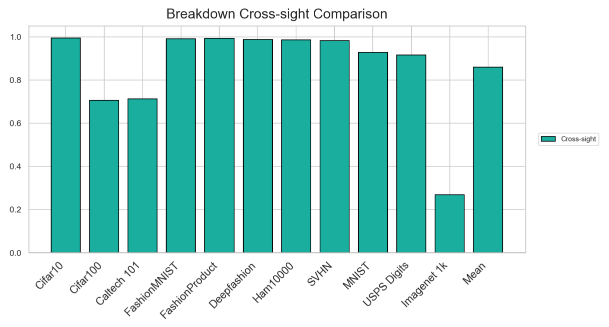
## 4.5 Experimental Results

We evaluate Cross-Select under four complementary generalization settings: (1) ModelSpider Comparison (known models, known datasets with unseen samples), (2) Blind Dataset (known models, new datasets), (3) Blind Models (new models, known datasets), and (4) Double Blind (new models, new datasets). All results are reported using the weighted Kendalls Tau metric $\tau_w$, which emphasizes correctness at the top of the predicted ranking.

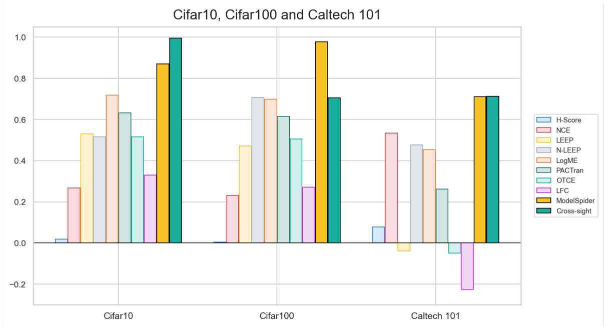
### 4.5.1 ModelSpider Comparison (Known Models, Known Datasets)



(a) Average Performance of Cross-Select compared to other methods and previous SOTA Model Spider



(b) Cross-Select Performance for known Datasets and Known Models



(c) Comparison Breakdown of Performance per Dataset

**Figure 4.4.** Comprehensive Model-Spider Comparison. (a) Detailed comparison of Cross-Select performance across all approaches. (b) Detailed breakdown per dataset. (c) Detailed performance comparison breakdown all using Weighted Kendall $\tau_w$.

This setting measures Cross-Selects ability to infer transferability among models that were fully observed during training, but evaluated on *unseen samples* from the same datasets. Cross-Select achieves consistently strong rank correlation across all datasets, often reaching values near $\tau_w = 0.86$. This performance surpasses Model Spider (reported $\tau_w = 0.75$) while using only *one image per class*, demonstrating far greater data efficiency compared to

Model Spiders ten images per class requirement. Cross-Select also substantially outperforms classical heuristics such as H-Score, NCE, LEEP, nLEEP, LogME, OTCE, and PACTran.

### 4.5.2 Blind Dataset (Known Models, New Datasets)

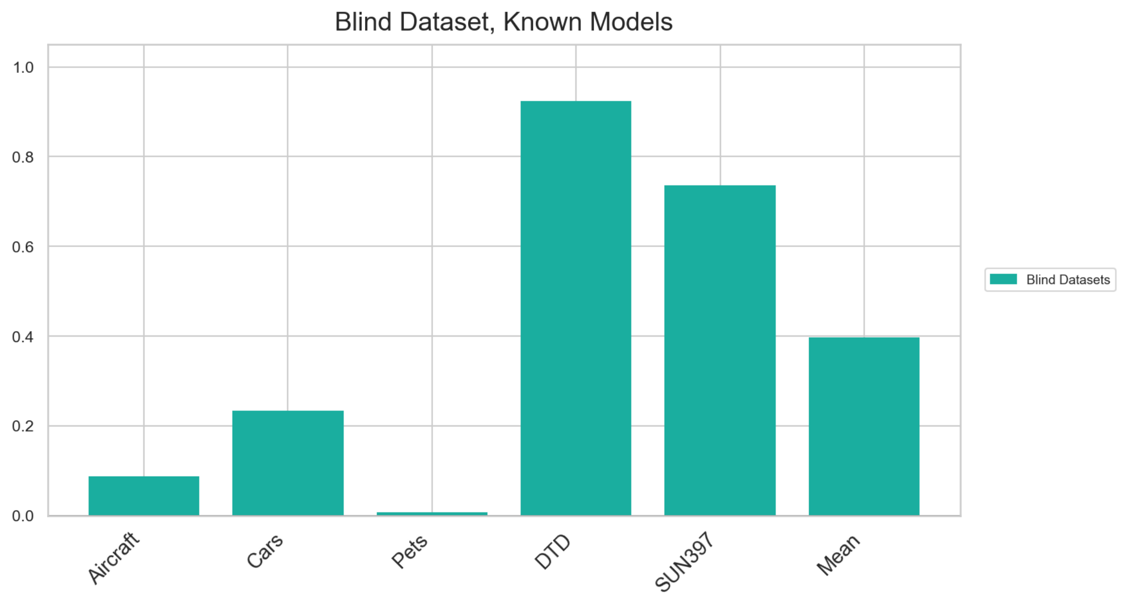


**Figure 4.5.** Blind Dataset Comparison (Known Models vs. Double Blind).

This scenario assesses Cross-Selects ability to generalize to entirely new datasets not used during training. Cross-Select retains positive rank correlation on all five blind datasets, with strong performance on texture-rich and scene-based datasets (e.g., DTD and SUN397). These results show that model tokens learned during training encode transferable structure that remains predictive even when the dataset distribution shifts.

In contrast, the double-blind condition (new datasets + new models) exhibits diminished correlation, indicating that dataset and model novelty compound the difficulty of the ranking task.

### 4.5.3 Blind Models (New Models, Known Datasets)

In this test, we evaluate whether Cross-Select can correctly rank models that were *never* seen during training. Performance drops substantially in this setting: although a few datasets produce mildly positive correlations, most hover around zero or slightly negative. This suggests that, while dataset conditioning transfers well, generalizing to unseen model architectures is challengingconfirming hypothesis H2 that generated tokens do not yet generalize robustly to unseen models.

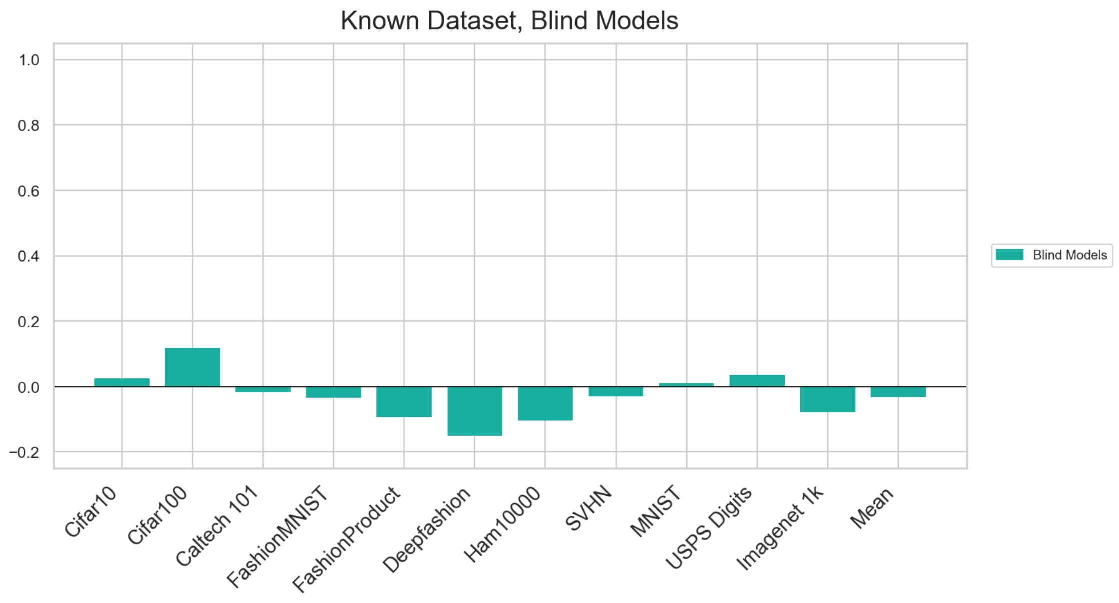


**Figure 4.6.** Blind Models on Known Datasets (Weighted Kendall $\tau_w$).

### 4.5.4 Double Blind (New Models, New Datasets)

The double-blind condition represents the most difficult generalization scenario: neither the datasets nor the models appear during training. Cross-Select produces weak but still non-degenerate correlations ($\tau_w$ values near 0 to 0.15). These results indicate partial transferability but highlight the need for improved model-token generalization mechanisms when both axes of variation shift simultaneously.

## 4.6 Limitations and Future Work

Although Cross-sight demonstrates the feasibility of learning datasetmodel compatibility through cross-attention and generated model tokens, several limitations constrain its current performance and generalizability.

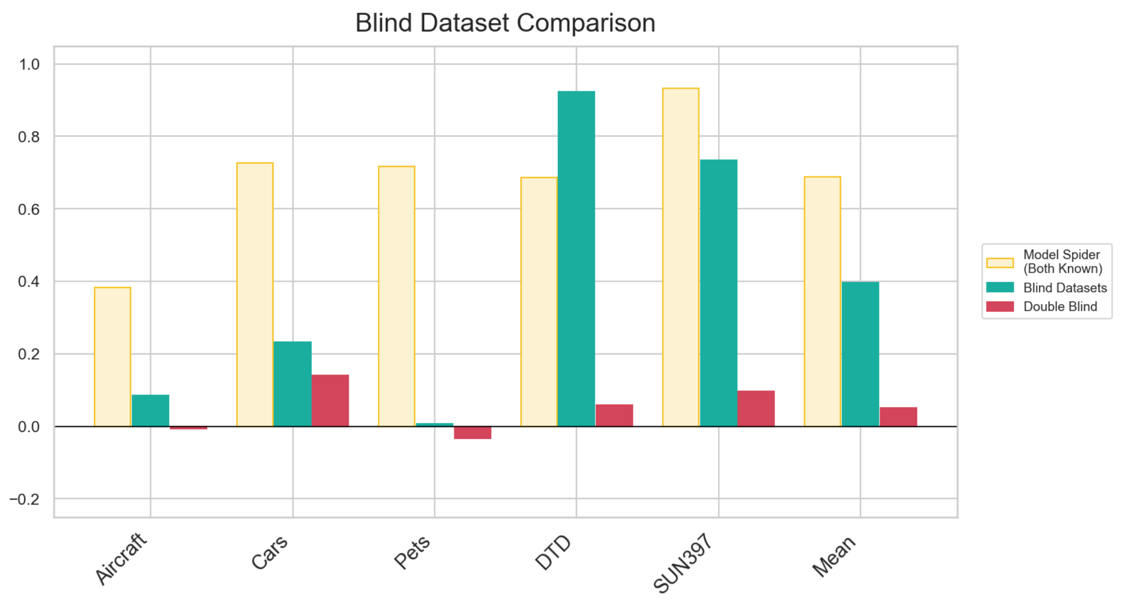


**Figure 4.7.** Double Blind Performance Across Novel Datasets (Weighted Kendall $\tau_w$).

First, the ranking space explored in this work is limited to a single compatibility dimension. Real-world PTM selection involves additional factorsincluding task constraints, hardware budgets, and deployment latency goalsthat are not explicitly modeled. Extending the system to support richer ranking objectives, such as those outlined in the proposed ModelSpider Fusion and ModelSpider Shadow pipelines, would allow a more comprehensive understanding of how different constraints influence model suitability.

Second, the current system relies on proxy signals rather than ground-truth fine-tuning performance. Because true rankings depend on actual downstream performance and hardware measurements, future work should involve building a dedicated dataset of PTMs fine-tuned across diverse tasks and hardware settings. Such a dataset would enable supervised training on realistic rankings and substantially improve the reliability of compatibility predictions.

Third, limitations in the ModelToVector pipeline restrict generalization to unseen models. The encoder captures only coarse architectural information and omits structural details, model-card metadata, and other semantically rich signals. Improving the model token generator — by integrating architecture encodings, metadata, and performing systematic ablations of the encoder — may enable more robust model-side representations.

Finally, the current training paradigm treats the encoder and transformer components independently. A more expressive framework would jointly optimize both components through hybrid learning, allowing the ranking loss to shape the model encoder directly. This could help mitigate the observed generalization gap for blind-model conditions.

## 4.7 Conclusions

In this work, we developed and evaluated Cross-Select, a framework for learning dataset-model compatibility through cross-attention and generated model tokens. Our experiments assessed two central hypotheses regarding (i) the benefits of cross-attention for alignment and (ii) the feasibility of replacing learned model tokens with generated ones.

The results provide partial confirmation of these hypotheses. Cross-attention substantially improved compatibility prediction for known modeldataset pairs and maintained reasonable performance on unseen datasets, indicating that it enables more expressive dataset-model alignment. Further, generated model tokens proved effective substitutes for learned tokens when operating within the distribution of known models, demonstrating that the token generator captures meaningful architectural semantics. However, the pipeline exhibited limited generalization to unseen models, with performance degrading sharply in blind-model conditions.

Overall, this study shows that Cross-Select can generalize across datasets but not across models, highlighting an important asymmetry in learned representations. These findings point to the need for richer model embeddings, broader architectural coverage, and more robust mechanisms for modeling model-level structure in future research.

# 5. SUMMARY AND CONCLUSIONS

This thesis examined the increasingly critical problem of selecting appropriate pre-trained models (PTMs) in a rapidly expanding and heterogeneous model ecosystem. As PTM repositories grow to include millions of models across modalities and architectures, practitioners face significant challenges in identifying models that align with their downstream tasks and deployment constraints. Existing approaches offer only partial solutions: heuristic matching and metadata-driven search overlook deeper datasetmodel compatibility, while state-of-the-art systems relying on fixed model zoos struggle to generalize to newly emerging PTMs. These limitations highlight the need for more principled, scalable, and context-aware recommendation techniques.

To address these challenges, this thesis contributed two complementary advances. First, it introduced a hardware-aware methodology for PTM selection, motivated by the unique constraints present in resource-limited environments such as IoT deployments. This vision outlined concrete metrics, design principles, and evaluation criteria for incorporating latency, memory, and energy considerations into automated model recommendation systems. Second, the thesis advanced the state of the art by developing a learning-based recommendation framework that leverages cross-attentionbased transformer architectures to model interactions between datasets and PTMs. This approach moves beyond heuristic similarity measures by directly learning representational alignment from prior evidence. Preliminary evaluations demonstrate that the proposed architecture matches or exceeds the performance of leading systems such as ModelSpider, while also offering improved generalization to unseen models.

Overall, this work deepens our understanding of the PTM selection problem and provides a foundation for more robust and scalable recommendation systems. The findings suggest several promising directions for future research, including incorporating multimodal datasets, extending the framework to model training dynamics, integrating cost-aware optimization, and exploring broader ecosystem-level signals such as model lineage and usage patterns. By enabling more effective PTM selection, the contributions of this thesis aim to improve the accessibility, reliability, and efficiency of machine learning development in both conventional and resource-constrained deployment environments.

# REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[2] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[3] 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 (NAACL-HLT)*, vol. 1, 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423.

[4] T. Brown et al., “Language models are few-shot learners,” in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 33, 2020, pp. 1877–1901.

[5] G. Hinton et al., “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups,” in *IEEE Signal Processing Magazine*, vol. 29, 2012, pp. 82–97. DOI: 10.1109/MSP.2012.2205597.

[6] A. Radford et al., “Learning transferable visual models from natural language supervision,” *arXiv preprint arXiv:2103.00020*, 2021.

[7] Y. Bengio, “Learning deep architectures for ai,” *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009. DOI: 10.1561/2200000006.

[8] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[9] V. Nair and G. E. Hinton, “Rectified linear units improve restricted Boltzmann machines,” in *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010, pp. 807–814.

[10] D. Hendrycks and K. Gimpel, *Gaussian error linear units (GELUs)*, 2016. arXiv: 1606.08415.

[11] Y. LeCun et al., “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. DOI: 10.1162/neco.1989.1.4.541.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, no. 6088, pp. 533–536, 1986. DOI: 10.1038/323533a0.

[13] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997. DOI: 10.1162/neco.1997.9.8.1735.

[14] K. Cho et al., “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1724–1734. DOI: 10.3115/v1/D14-1179.

[15] A. Vaswani et al., “Attention is all you need,” in *Advances in Neural Information Processing Systems*, 2017, pp. 5998–6008.

[16] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” *OpenAI Technical Report*, 2019.

[17] A. Dosovitskiy et al., “An image is worth 16x16 words: Transformers for image recognition at scale,” in *International Conference on Learning Representations (ICLR)*, 2021.

[18] L. Bottou, “Large-scale machine learning with stochastic gradient descent,” in *Proceedings of COMPSTAT’2010*, Springer, 2010, pp. 177–186. DOI: 10.1007/978-3-7908-2604-3_16.

[19] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” in *International Conference on Learning Representations (ICLR)*, 2015.

[20] T. Tieleman and G. Hinton, *Lecture 6.5 – RMSProp: Divide the gradient by a running average of its recent magnitude*, COURSERA: Neural Networks for Machine Learning, 2012.

[21] I. Loshchilov and F. Hutter, “Decoupled weight decay regularization,” in *International Conference on Learning Representations (ICLR)*, 2019.

[22] E. Strubell, A. Ganesh, and A. McCallum, “Energy and policy considerations for deep learning in NLP,” in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2019, pp. 3645–3650. DOI: 10.18653/v1/P19-1355.

[23] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, “On the dangers of stochastic parrots: Can language models be too big?” In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 2021, pp. 610–623. DOI: 10.1145/3442188.3445922.

[24] S. J. Pan and Q. Yang, “A survey on transfer learning,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010. DOI: 10.1109/TKDE.2009.191.

[25] K. Weiss, T. M. Khoshgoftaar, and D. Wang, “A survey of transfer learning,” *Journal of Big Data*, vol. 3, no. 1, pp. 1–40, 2016. DOI: 10.1186/s40537-016-0043-6.

[26] F. Zhuang et al., “A comprehensive survey on transfer learning,” *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2021. DOI: 10.1109/JPROC.2020.3004555.

[27] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, “How transferable are features in deep neural networks?” In *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 27, 2014, pp. 3320–3328.

[28] Z. Alyafeai, M. S. AlShaibani, and I. Ahmad, *A survey on transfer learning in natural language processing*, May 2020. DOI: 10.48550/arXiv.2007.04239. arXiv: 2007.04239.

[29] J. Donahue et al., “DeCAF: A deep convolutional activation feature for generic visual recognition,” in *International Conference on Machine Learning (ICML)*, PMLR, 2014, pp. 647–655.

[30] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, “CNN features off-the-shelf: An astounding baseline for recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014, pp. 806–813. DOI: 10.1109/CVPRW.2014.131.

[31] J. Howard and S. Ruder, “Universal language model fine-tuning for text classification,” in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, vol. 1, 2018, pp. 328–339. DOI: 10.18653/v1/P18-1031.

[32] Y. Ganin et al., “Domain-adversarial training of neural networks,” *Journal of Machine Learning Research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[33] M. Long, Y. Cao, J. Wang, and M. I. Jordan, “Learning transferable features with deep adaptation networks,” in *International Conference on Machine Learning (ICML)*, PMLR, 2015, pp. 97–105.

[34] T. Zhang, Y. Shu, X. Chen, Y. Long, C. Guo, and B. Yang, “Assessing pre-trained models for transfer learning through distribution of spectral components,” *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 21, pp. 22 560–22 568, Apr. 2025. DOI: 10.1609/aaai.v39i21.34414.

[35] A. T. Tran, C. V. Luong, T.-D. Nghiem, and T. Nguyen, “Transferability and hardness of supervised classification tasks,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1395–1405. DOI: 10.1109/ICCV.2019.00148.

[36] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese, “Taskonomy: Disentangling task transfer learning,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 3712–3722. DOI: 10.1109/CVPR.2018.00391.

[37] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A large-scale hierarchical image database,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[38] S. Kornblith, J. Shlens, and Q. V. Le, “Do better ImageNet models transfer better?” In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 2661–2671. DOI: 10.1109/CVPR.2019.00277.

[39] K. He, R. Girshick, and P. Dollár, “Rethinking ImageNet pre-training,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 4918–4927. DOI: 10.1109/ICCV.2019.00502.

[40] Z.-H. Zhou, “Learnware: On the future of machine learning,” *Frontiers of Computer Science*, vol. 10, no. 4, pp. 589–590, 2016. DOI: 10.1007/s11704-016-6906-3.

[41] L.-Z. Guo, Z. Zhou, Y.-F. Li, and Z.-H. Zhou, “Identifying useful learnwares for heterogeneous label spaces,” in *Proceedings of the 40th International Conference on Machine Learning*, PMLR, Jul. 2023, pp. 12 122–12 131. [Online]. Available: https://proceedings.mlr.press/v202/guo23l.html.

[42] Y.-K. Zhang, T.-J. Shen, S.-J. Yu, Y. Li, Y.-F. Li, and Z.-H. Zhou, "Model spider: Learning to rank pre-trained models efficiently," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023, pp. 1–14.

[43] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," *Journal of Machine Learning Research*, vol. 21, no. 140, pp. 1–67, 2020.

[44] T. Wolf et al., "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020, pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6.

[45] M. Abadi et al., *TensorFlow: Large-scale machine learning on heterogeneous systems*, 2015. [Online]. Available: https://www.tensorflow.org/.

[46] A. Paszke et al., "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 32, 2019, pp. 8024–8035.

[47] P. Liu, Y. Zhang, Y. Zhou, Y. Liu, and Y. Liu, *PTMPicker: Facilitating efficient pretrained model selection for application developers*, Aug. 2025. DOI: 10.48550/arXiv.2508.11179. arXiv: 2508.11179.

[48] F. Meng et al., "Foundation model is efficient multimodal multitask model selector," *Advances in Neural Information Processing Systems*, vol. 36, pp. 33 065–33 094, Dec. 2023.

[49] P. Tan, H.-T. Liu, Z.-H. Tan, and Z.-H. Zhou, "Handling learnwares from heterogeneous feature spaces with explicit label exploitation," *Advances in Neural Information Processing Systems*, vol. 37, pp. 12 767–12 795, Dec. 2024.

[50] R. Schwartz, J. Dodge, N. A. Smith, and O. Etzioni, "Green AI," in *Communications of the ACM*, vol. 63, ACM, 2020, pp. 54–63. DOI: 10.1145/3381831.

[51] C.-J. Wu et al., "Sustainable AI: Environmental implications, challenges and opportunities," *Proceedings of Machine Learning and Systems*, vol. 4, pp. 795–813, 2022.

[52] B. R. Team, *Improving LLMs for recommendation with out-of-vocabulary tokens*, Accessed: Sept. 21, 2025, Sep. 2025. [Online]. Available: https://bytez.com/docs/icml/44677/paper.

[53] K. Shah, C. Sheth, and N. Doshi, “A survey on IoT-based smart carsu, their functionalities and challenges,” in *Procedia Computer Science*, ser. The 13th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN) / The 12th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2022) / Affiliated Workshops, vol. 210, Jan. 1, 2022, pp. 295–300. DOI: 10.1016/j.procs.2022.10.153.

[54] K. S. Pratyush Reddy, Y. M. Roopa, K. Rajeev L.N., and N. S. Nandan, “IoT based smart agriculture using machine learning,” in *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, Jul. 2020, pp. 130–134. DOI: 10.1109/ICIRCA48905.2020.9183373.

[55] P. J. Rani, J. Bakthakumar, B. P. Kumaar, U. P. Kumaar, and S. Kumar, “Voice controlled home automation system using natural language processing (NLP) and internet of things (IoT),” in *2017 Third International Conference on Science Technology Engineering & Management (ICONSTEM)*, Mar. 2017, pp. 368–373. DOI: 10.1109/ICONSTEM.2017.8261311.

[56] M. Almeida, S. Laskaridis, A. Mehrotra, L. Dudziak, I. Leontiadis, and N. D. Lane, “Smart at what cost? characterising mobile deep neural networks in the wild,” in *Proceedings of the 21st ACM Internet Measurement Conference*, ser. IMC ’21, Virtual Event: Association for Computing Machinery, 2021, pp. 658–672, ISBN: 9781450391290. DOI: 10.1145/3487552.3487863. [Online]. Available: https://doi.org/10.1145/3487552.3487863.

[57] W. Jiang et al., “An empirical study of pre-trained model reuse in the hugging face deep learning model registry,” in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, ISSN: 1558-1225, May 2023, pp. 2463–2475. DOI: 10.1109/ICSE48619.2023.00206.

[58] N. K. Gopalakrishna, D. Anandayuvaraj, A. Detti, F. L. Bland, S. Rahaman, and J. C. Davis, “"if security is required": Engineering and security practices for machine learning-based IoT devices,” in *Proceedings of the 4th International Workshop on Software Engineering Research and Practice for the IoT*, Pittsburgh Pennsylvania: ACM, May 19, 2022, pp. 1–8, ISBN: 978-1-4503-9332-4. DOI: 10.1145/3528227.3528565.

[59] A. T. Tran, C. V. Nguyen, and T. Hassner, “Transferability and hardness of supervised classification tasks,” in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 1395–1405.

[60] Y. Bao et al., “An information-theoretic approach to transferability in task transfer learning,” in *2019 IEEE International Conference on Image Processing (ICIP)*, ISSN: 2381-8549, Sep. 2019, pp. 2309–2313. DOI: 10.1109/ICIP.2019.8803726.

[61] 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*, ISSN: 2640-3498, PMLR, Nov. 21, 2020, pp. 7294–7305.

[62] Y. Li et al., “Ranking neural checkpoints,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 2663–2673.

[63] 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*, ISSN: 2640-3498, PMLR, Jul. 1, 2021, pp. 12 133–12 143.

[64] N. Ding, X. Chen, T. Levinboim, S. Changpinyo, and R. Soricut, “PACTran: PAC-bayesian metrics for estimating the transferability of pretrained models to classification tasks,” in *Computer Vision  ECCV 2022*, ISSN: 1611-3349, Springer, Cham, 2022, pp. 252–268, ISBN: 978-3-031-19830-4. DOI: 10.1007/978-3-031-19830-4_15.

[65] M. Pándy, A. Agostinelli, J. Uijlings, V. Ferrari, and T. Mensink, “Transferability estimation using bhattacharyya class separability,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 9172–9182.

[66] A. Deshpande et al., “A linearized framework and a new benchmark for model selection for fine-tuning,” no. arXiv:2102.00084, Jan. 29, 2021. DOI: 10.48550/arXiv.2102.00084. arXiv: 2102.00084.

[67] Y.-K. Zhang, T.-J. Huang, Y.-X. Ding, D.-C. Zhan, and H.-J. Ye, “Model spider: Learning to rank pre-trained models efficiently,” in *Advances in Neural Information Processing Systems*, vol. 36, Dec. 15, 2023, pp. 13 692–13 719.

[68] F. Meng et al., “Foundation model is efficient multimodal multitask model selector,” *Advances in Neural Information Processing Systems*, vol. 36, pp. 33 065–33 094, Dec. 15, 2023.

[69] J. Bai, S. Wu, J. Song, J. Zhao, and G. Chen, “Pre-trained model recommendation for downstream fine-tuning,” *arXiv*, 2024. DOI: 10.48550/ARXIV.2403.06382.

[70] Y. Tan, Y. Li, and S.-L. Huang, “OTCE: A transferability metric for cross-domain cross-task representations,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 15 779–15 788.

[71] P. M. Sánchez Sánchez, J. M. Jorquera Valero, A. Huertas Celdrán, G. Bovet, M. Gil Pérez, and G. Martínez Pérez, “LwHBench: A low-level hardware component benchmark and dataset for single board computers,” *Internet of Things*, vol. 22, p. 100 764, Jul. 1, 2023, ISSN: 2542-6605. DOI: 10.1016/j.iot.2023.100764.

[72] M. F. A. Sayeedi, J. F. Deepti, A. M. I. M. Osmani, T. Rahman, S. S. Islam, and M. M. Islam, “A comparative analysis for optimizing machine learning model deployment in IoT devices,” *Applied Sciences*, vol. 14, no. 13, p. 5459, Jan. 2024, Number: 13 Publisher: Multidisciplinary Digital Publishing Institute, ISSN: 2076-3417. DOI: 10.3390/app14135459.

[73] J. C. Davis, P. Jajal, W. Jiang, T. R. Schorlemmer, N. Synovic, and G. K. Thiruvathukal, “Reusing deep learning models: Challenges and directions in software engineering,” in *2023 IEEE John Vincent Atanasoff International Symposium on Modern Computing (JVA)*, Jul. 2023, pp. 17–30. DOI: 10.1109/JVA60410.2023.00015.

[74] P. Jajal et al., “Interoperability in deep learning: A user survey and failure analysis of onnx model converters,” in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2024, pp. 1466–1478.

[75] W. Jiang et al., “Challenges and practices of deep learning model reengineering: A case study on computer vision,” *Empirical Software Engineering*, Aug. 25, 2023. DOI: 10.48550/arXiv.2303.07476.

[76] A. Copeland, “A reasonable social welfare function,” in *Seminar on Applications of Mathematics to Social Sciences*, Mimeographed Notes, University of Michigan, Ann Arbor, 1951.

[77] D. G. Saari and V. R. Merlin, “The copeland method: I.: Relationships and the dictionary,” *Economic Theory*, vol. 8, no. 1, pp. 51–76, Feb. 1996, ISSN: 0938-2259, 1432-0479. DOI: 10.1007/BF01212012.

[78] H. Benmeziane, K. E. Maghraoui, H. Ouarnoughi, S. Niar, M. Wistuba, and N. Wang, “A comprehensive survey on hardware-aware neural architecture search,” *ArXiv*, vol. abs/2101.09336 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:231699126.

[79] M. Tan et al., " MnasNet: Platform-Aware Neural Architecture Search for Mobile," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2019. DOI: 10.1109/CVPR.2019.00293. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2019.00293.

[80] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y. Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, ISSN: 2640-3498, PMLR, Apr. 10, 2017, pp. 1273–1282.

[81] A. Makhshari and A. Mesbah, "Iot bugs and development challenges," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, IEEE, 2021, pp. 460–472.

[82] J. Garcia et al., "A comprehensive study of autonomous vehicle bugs," in *Proceedings of the ACM/IEEE 42nd international conference on software engineering*, 2020, pp. 385–396.

[83] V. Banna et al., "An experience report on machine learning reproducibility: Guidance for practitioners and tensorflow model garden contributors," *arXiv preprint arXiv:2107.00821*, 2021.

[84] T. Lin, Y. Wang, X. Liu, and X. Qiu, "A survey of transformers," *AI Open*, vol. 3, pp. 111–132, 2022. DOI: 10.1016/j.aiopen.2022.10.001.

[85] L. Tunstall, L. von Werra, and T. Wolf, *Natural Language Processing with Transformers: Building Language Applications with Hugging Face*. O'Reilly Media, 2022, ISBN: 978-1-098-10324-3.

[86] S. Khan, M. Naseer, M. Hayat, S. W. Zamir, F. S. Khan, and M. Shah, "Transformers in vision: A survey," *ACM Computing Surveys*, vol. 54, no. 10s, pp. 1–41, 2022. DOI: 10.1145/3505244.

[87] M. Phuong and M. Hutter, "Formal algorithms for transformers," in *arXiv preprint arXiv:2207.09238*, 2022. arXiv: 2207.09238.

[88] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, "Megatron-LM: Training multi-billion parameter language models using model parallelism," in *arXiv preprint arXiv:1909.08053*, 2019. arXiv: 1909.08053.

[89] Y. Tay, M. Dehghani, D. Bahri, and D. Metzler, “Efficient transformers: A survey,” *ACM Computing Surveys*, vol. 55, no. 6, pp. 1–28, 2022. DOI: 10.1145/3530811.

[90] M. G. Kendall, “A new measure of rank correlation,” *Biometrika*, vol. 30, no. 1/2, pp. 81–93, 1938. DOI: 10.2307/2332226.

[91] A. Agresti, *Analysis of Ordinal Categorical Data*, 2nd. John Wiley & Sons, 2010, ISBN: 978-0-470-08289-8. DOI: 10.1002/9780470594001.

[92] S. Vigna, “A weighted correlation index for rankings with ties,” *Proceedings of the 24th International Conference on World Wide Web*, pp. 1166–1176, 2015. DOI: 10.1145/2736277.2741088.

[93] R. Kumar and S. Vassilvitskii, “Generalized distances between rankings,” in *Proceedings of the 19th International Conference on World Wide Web*, ACM, 2010, pp. 571–580. DOI: 10.1145/1772690.1772749.