

Article

RegRes: An Exploratory Study of Cross-Platform and Cross-Code Performance Prediction Using Regression Models

Jie Li ¹, Gui Zhao ^{1,*} , Biqing Zeng ¹, Jingxin Liu ² and Fangjie Chen ²

¹ School of Artificial Intelligence, South China Normal University, Foshan 528225, China; lijie@scnu.edu.cn (J.L.); zengbiqing@scnu.edu.cn (B.Z.)

² School of Mathematics and Physics Sciences, RI-IM·AI *, Chongqing University of Science and Technology, Chongqing 401331, China; jingxin_liu127@cqust.edu.cn (J.L.); 2025212023@cqust.edu.cn (F.C.)

* Correspondence: 2023024333@m.scnu.edu.cn

Abstract

In the era of heterogeneous computing environments, including diverse hardware platforms and programming paradigms, accurate performance prediction of software applications is essential for efficient resource allocation, cost optimization, and informed deployment decisions. However, traditional methods often require platform-specific measurements, which are resource-intensive and limited by data scarcity in low-performance or novel code contexts. This exploratory study addresses these challenges by leveraging transfer learning with regression models to enable cross-platform (from Intel x86 to ARM M1) and cross-code performance predictions. Using the Renaissance benchmark suite (renaissance-gpl-0.15.0.jar), we systematically evaluate eight traditional machine learning models and a deep neural network across scenarios of data transfer and fine-tuning. Key findings demonstrate that transfer learning significantly improves prediction accuracy, with tree-based models like Extra Trees achieving high R^2 scores (0.92) and outperforming DNNs in robustness, particularly under noisy or data-scarce conditions. The study provides empirical insights into model effectiveness, highlights the superiority of transfer settings, and offers practical guidance for software engineers to reduce measurement overheads and enhance optimization processes.

Keywords: cross-platform prediction; cross-code prediction; transfer learning; regression models; deep neural networks

1. Introduction

Accurate performance prediction is a fundamental challenge in software engineering [1–3]. With the rise in cloud services [4,5], mobile devices [6,7], and heterogeneous computing platforms [8–10], modern applications are increasingly required to execute efficiently across diverse hardware environments. In cloud environments, accurate prediction enables dynamic resource allocation to minimize costs while meeting service-level agreements. On mobile devices, predicting energy consumption is crucial for extending battery life. For practitioners, the ability to predict execution time and resource consumption is critical to support deployment decisions, resource allocation, and cost optimization. However, building reliable performance models remains difficult. Traditional approaches [11,12] are typically platform-specific, requiring extensive measurement and model calibration for each new environment. This process is costly and often infeasible when the target platform has limited computational resources or measurement opportunities. Moreover, performance



Received: 25 November 2025

Revised: 30 December 2025

Accepted: 12 January 2026

Published: 15 January 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

is not only hardware-dependent but also influenced by code characteristics such as concurrency patterns, memory access behavior, and computational complexity. These factors make cross-platform and cross-code performance prediction particularly challenging.

While existing performance prediction techniques, such as analytical modeling [13–15] and machine learning [16–19], offer valuable insights, they often rely on accurate data and may not generalize well across different hardware architectures or code types. Similarly, while transfer learning has shown promise in software engineering for tasks such as defect detection and cross-platform adaptation, existing studies have limitations in fully considering the impact of code types, differences between codes, and the challenges of cross-platform performance prediction. Despite the growing interest in cross-platform adaptation, there is a lack of comprehensive empirical studies that systematically compare different machine learning models for cross-platform performance prediction [20,21], particularly considering the combined effects of hardware architectures and code types. Many existing studies focus on specific application domains or code types, limiting the generalizability of their findings. There is a need for a more systematic investigation of cross-code performance prediction [22] across a wider range of software benchmarks. Furthermore, the performance of deep learning models in cross-platform and cross-code performance prediction scenarios, especially in comparison to traditional machine learning models, remains relatively unexplored. Existing work often lacks a strong focus on the practical implications for software engineers, such as providing guidance on model selection and reducing measurement costs.

Therefore, in this study, we systematically compared eight traditional machine learning models and one deep neural network for cross-platform and cross-code performance prediction. Using a diverse set of software benchmarks, we evaluated the effectiveness of transfer learning in cross-platform and cross-code scenarios. In the context of cross-platform and cross-code performance prediction, we conducted a detailed analysis of the strengths and weaknesses of different machine learning models, including deep neural networks. However, in terms of data scarcity and heterogeneity hindering performance prediction, we face three challenging issues, as follows.

- Q1: In cross-platform prediction, can data from high-performance computing environments (e.g., Intel x86 architecture) enhance the predictive performance on low-performance environments (e.g., APPLE ARM M1 architecture)?
- Q2: In cross-code prediction, can runtime data and features extracted from frequently analyzed codebases be effectively transferred to predict the performance of rarely examined codebases?
- Q3: How do deep neural networks (DNNs) measure up against traditional machine learning (ML) methods in scenarios demanding transfer learning?

To address the aforementioned issues, we presented RegRes, an exploratory framework that systematically compares traditional machine learning algorithms and deep neural networks for cross-platform and cross-code performance prediction. Our study uses 19 representative benchmarks executed on Intel x86 and ARM M1 processors. The experimental results show that while transfer learning is effective in both scenarios, traditional machine learning models generally outperform DNNs, especially in data-scarce conditions. The contributions of this work are threefold:

1. We proposed a novel application of a regression model in software engineering performance prediction, and introduced a unique framework (RegRes), which can achieve efficient cross-platform and cross-code prediction in data-scarce scenarios, and solves a key gap in the existing platform-specific methods.
2. We innovatively fine-tuned regression models to achieve superior performance enhancements, and conducted the first systematic comparative evaluation of eight

traditional machine learning models against one deep neural network in cross-code transfer learning settings, revealing new insights into model adaptability across heterogeneous environments.

3. The experimental results demonstrate a breakthrough in achieving robust performance on multiple platforms with limited data volumes—nearly reaching 80% of the baseline’s effectiveness.

The remainder of this paper is organized as follows. Section 2 reviews related work on performance prediction and transfer learning. Section 3 presents the system overview and method. Section 4 reports the experimental evaluation, followed by Section 5, which discusses the analysis of results. Section 6 concludes the paper.

2. Related Works

In this section, we will briefly introduce performance prediction and the application of transfer learning in software engineering, as well as the specific problems we aim to address.

2.1. Performance Prediction Techniques

2.1.1. Analytical Modeling

Seung-Jun Shin et al. [23] proposed a model that uses big data analysis to predict electricity consumption, with the aim of achieving more accurate and timely decision making by analyzing vast manufacturing data. Using a data-driven analytical modeling approach, the model can identify applicable data patterns from real-world data, leading to more precise predictions of electricity usage. The advantage of this approach is its ability to handle large-scale data, making it suitable for complex manufacturing environments. However, its accuracy heavily depends on the quality and completeness of the data; if the data are inaccurate or incomplete, the prediction results may be unreliable. Daya Atapattu et al. [13] integrated analytical models into an interactive software tool for performance prediction. In their research, they also proposed a simple cache miss model to forecast performance degradation caused by data access. This model considers factors, such as cache size and access patterns, providing an estimate of the cache miss rate. The benefit of this approach is that the analytical model offers better insight into the performance bottlenecks of programs. However, the cache miss model presented is relatively simple and may not accurately predict the performance of more complex programs. Additionally, the tool is primarily designed for the Alliant FX/8 architecture and may not be suitable for other mainstream parallel systems.

2.1.2. Statistical Modeling

Janne Riihijärvi et al. [6] used machine learning methods to predict the performance of mobile cellular networks. They employed real data as the dataset and addressed the challenge of spatial and temporal prediction in the real world, which is difficult to measure. However, since signal processing data can be affected by hardware and environmental factors, the data may suffer from various impairments. Additionally, differences between hardware architectures lead to poor model scalability. Gopinath Chennupati et al. [24] used machine learning and regression techniques to construct a scalable performance prediction model for scientific codes. By analyzing the basic block structure and memory reuse of code segments, they aimed to predict execution time. However, because memory performance is significantly influenced by the operating system’s allocation, the model still exhibits limited scalability in certain scenarios. Diego Didona et al. [25] explored ways to enhance the robustness of performance prediction by integrating analytical modeling (AM) and machine learning (ML). Their work introduced three methodologies: K-Nearest Neighbors

(KNN), which assesses the accuracy of the AM model and ML methods within the feature space during the learning process and selects the optimal prediction approach using the KNN algorithm; Hybrid Boosting (HyBoost), which employs a chain of ML algorithms to learn the residual errors of AM and makes performance predictions by utilizing AM outputs in conjunction with error correction functions; and Prediction Region (PR), which focuses specifically on predicting areas of the feature space where AM is inaccurate. This method involves using classifiers to identify regions with significant AM errors and employing regressors for predictions in those areas. The main advantage of combining AM and ML is the construction of more robust performance models, reducing dependence on specific technologies or datasets. However, no single hybrid technique consistently outperforms others across all scenarios considered, and the performance of hybrid methods is largely contingent on the quality of the AM employed.

2.2. Transfer Learning in Software Engineering

Given current development trends, the demand for high-quality software has increased year over year. However, the lack of available data for training and testing makes it difficult to ensure the performance and reliability of software. The idea of transfer learning provides new perspectives for improving software reusability and scalability [26].

2.2.1. Cross-Platform Adaptation

In the area of cross-platform development, we summarize some studies from the traditional software engineering field. Rajat Kumar et al. [27] demonstrates the performance differences in classic algorithms across various platforms and employs transfer learning to predict how these codes perform on different hardware platforms. However, their experiments did not fully consider the impact of code types and differences between codes, which significantly limits the applicability to particular algorithms. Ma et al. [28] introduced transfer learning into defect detection for cross-company software, addressing the limitation of traditional defect prediction models that mostly rely on internal company data. In real-world scenarios, many enterprises face challenges such as high labeling costs and the novelty of project domains, leading to a lack of local training data. However, they did not explore the potential of combining transfer learning with other classifiers, such as decision trees and SVMs.

2.2.2. Cross-System Adaptation

There are also some studies in the area of cross-system development. Chen et al. [29] address the problem of cross-system log anomaly detection by employing transfer learning. They extract semantic information from logs and utilize an innovative LSTM model to enable noise-resistant cross-system knowledge transfer. Lov Kumar et al. [30] tackles the issue of data scarcity across systems and projects by evaluating the performance of eight machine learning classifiers on both within-project and cross-platform data.

2.3. Code Quality Maintenance Tools and Methods

2.3.1. Static Code Quality Tools

Representative static code analysis tools such as SonarQube (<https://www.sonarsource.com/>) [31–33] evaluate code quality through the “Seven Axes of Quality” (design architecture, code duplication, comment completeness, unit test effectiveness, complexity, potential bugs, and coding standards compliance). It integrates with build tools (e.g., Ant (<https://ant.apache.org/>), Maven (<https://maven.apache.org/>)) and CI/CD pipelines (e.g., Jenkins (<https://www.jenkins.io/>)), rule-based defect analysis, and static metric calculation to provide real-time quality feedback during the development workflow, enabling early identification of issues like code redundancy and coding standard violations.

These tools focus solely on static code analysis and lack the capability to predict runtime performance across heterogeneous hardware platforms. They neither support resource allocation optimization nor address dynamic performance fluctuations. Therefore, we propose the RegRes framework, using transfer learning to predict execution time in cross-platform environments. It provides data-driven support for software development in resource-constrained scenarios, making up for the shortcomings of static code quality tools in performance optimization.

2.3.2. Automated Unit Testing

Automated unit testing, like JUnit [34], verifies the correctness of minimal code units (e.g., function) to find early defects. The key methodology includes designing test cases following the FIRST-AIR principle, integrating test development with production code engineering via Test-Driven Development (TDD), and improving code maintainability through test patterns. Traditional unit testing demands substantial development and maintenance efforts for platform-specific test suites. It only validates functional correctness without assessing cross-platform performance differences, and exhibits poor applicability in data-scarce scenarios (e.g., novel codebases or hardware). Our model RegRes uses cross-code transfer learning to address the issue of data sparse, reducing the cost of test data collection. Additionally, its performance prediction capability guides the prioritization of unit testing and enhances testing efficiency.

2.3.3. Software Maintainability Models

Early software maintainability models [35] track code health through quantitative metrics such as code complexity, coupling, and cohesion. These models enable teams to identify code health trends and prevent technical difficult accumulation. Follow-up studies [36] have highlighted the limitations of single maintainability indices and proposed multi-dimensional models for more accurate maintenance difficulty assessment. Existing models focus on static code attributes (e.g., lines of code, comment rate) and fail to correlate the code's dynamic performance. They cannot address maintenance challenges in heterogeneous hardware environments. Therefore, we extend the maintainability assessment to the performance dimension by predicting cross-platform and cross-code execution time. Furthermore, the transfer learning capability enables the rapid establishment of performance maintainability correlation models for hardware platforms in a data-scarce scenario.

3. Model Design

3.1. Overview of the RegRes Framework

Considering the prediction of performance in different scenarios, in the cross-platform scenario, we compare the performance of different machine learning models; in the cross-code scenario, we compare the transfer learning performance of different codes and the performance of different machine learning models. Machine learning improves the prediction performance by training on a large amount of data to gain experience. However, we only have a limited code type to train. This affects prediction performance when the prediction only acquires a small amount of experience, such as over-fitting. RegRes is proposed to solve the problem that machine learning is hindered in the presence of limited data, and the key to solving the problem is to use transfer learning. Therefore, RegRes can generally be divided into two categories: one is to utilize models trained on high-performance platforms and then fine-tune them to predict performance on low-performance platforms, while the other is to utilize models trained on other code types and fine-tune them to predict the performance of the target code.

The RegRes framework is shown in Figure 1. The model starts with data collection and preprocessing, and uses data collected from 19 representative benchmarks as the basis. Then it entered the model training and fine-tuning stage, focused on the optimization of the regression model, and compared it with a deep neural network. Next is the transfer learning application, which realizes cross-platform and cross-code prediction by training the model in the source domain (e.g., a high-performance host, such as Intel x86) and transferring its knowledge to the target domain (e.g., a low-performance host, such as Apple ARM M1) or different code types. In the evaluation stage, the performance of different models is compared and evaluated. Finally, the hypothesis is verified through the result analysis, providing practical insights for software engineers. The framework emphasizes efficient prediction in the context of data scarcity, highlighting the advantages of transfer learning and regression models.

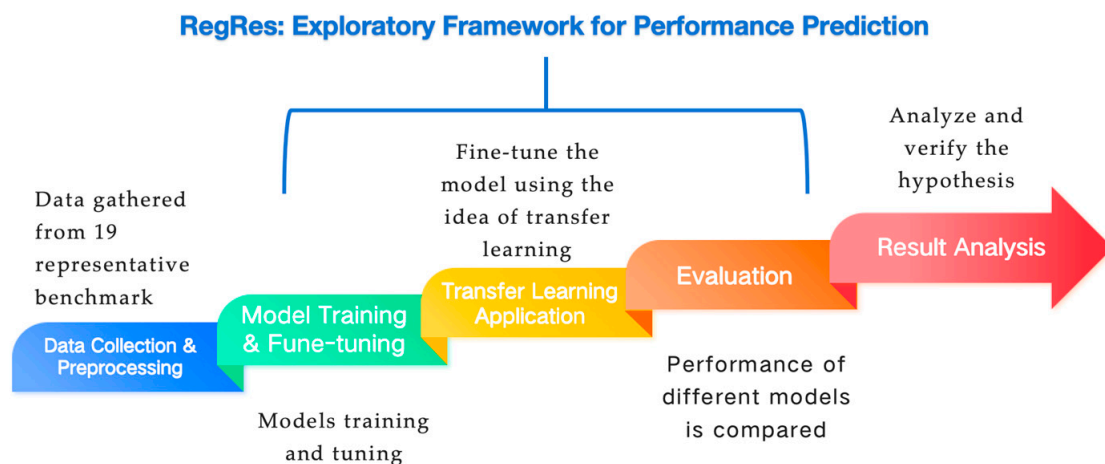


Figure 1. Overview of RegRes.

3.2. Research Methodology

In the research methodology, we first expound on data preparation and the composition of the dataset; then introduce the training and evaluation of the model, and finally describe the general workflow of RegRes in the form of pseudo-code.

3.2.1. Data Preparation and Preprocessing

We construct the dataset for model training based on the Renaissance benchmark [37] (see in Appendix A), which contains a total of 1231 data points. The distribution of data points for each benchmark is as follows: akka-uct (48), db-shootout (31), dotty (74), dummy-empty (38), dummy-failing (2), dummy-param (38), dummy-teardown-failing (36), dummy-validation-failing (2), finagle-chirper (180), finagle-http (22), fj-kmeans (60), future-genetic (100), mnemonics (32), par-mnemonics (32), philosophers (60), rx-scrabble (158), scala-kmeans (100), scala-stm-bench7 (120), and scrabble (98). To ensure stable measurements, each benchmark is executed multiple times to reduce noise. We impute any missing values with the median of the corresponding features. Static code features are extracted from the source code using an Abstract Syntax Tree (AST) parser, specifically leveraging the open-source AST explorer (<https://astexplorer.net/>) [38] and Clang (17.0.6) [39]. The extract static features help capture potential runtime influences on performance, thereby facilitating our transfer learning tasks.

To explore the relationship within the datasets (in Figure 2), we define the execution time that we aim to predict as Y_i , and we represent each training program as a feature vector X_i (see Equation (1)). The feature vector comprises features ranging from f_1 to f_n , encapsulating runtime statistics for each benchmark (e.g., execution duration, maximum

and minimum overhead times) along with machine physical information and code features. Using transfer learning integrated with various machine learning methods, we aim to learn the mapping relationship between X_i and Y_i as represented by Equation.

$$X_i = [f_1, f_2, \dots, f_n, \text{runtime statistics and features}] \quad (1)$$

$$Y_i = f(X_i) \quad (2)$$

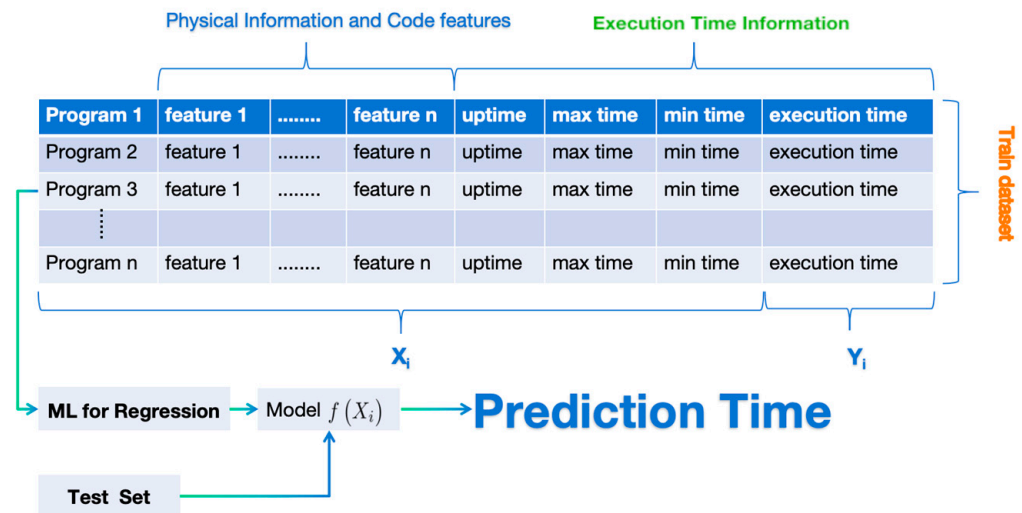


Figure 2. Dataset model.

3.2.2. Model Training and Comparison

We evaluate eight regression models (summarized in Table 1) alongside one deep neural network, implementing all models using scikit-learn and TensorFlow. The dataset is divided using an 80/20 train-test split, and each experiment is repeated five times with different random seeds. For transfer learning, initial model training occurs on the source domain (from a different platform or code type), followed by fine-tuning or direct evaluation on the target domain. To systematically evaluate these approaches, we consider the following scenarios in our model design.

Table 1. Regression model selection.

Model	Alias	Important Param
Random Forest Model	rf	max_depth, min_samples_split
Decision Tree Model	dt	max_depth, min_samples_split
K-Nearest Neighbor Model	knn	n_neighbors, weights
Gaussian Process Model	gpr	kernel, alpha
Support Vector Model	svr	C, kernel, gamma
Ridge Model	rr	alpha
Extra Trees Model	etr	n_estimators, max_features
Gradient Boosting Model	gbr	n_estimators, learning_rate

1. **Baseline:** This model is trained solely on the target benchmark data. Cross-validation (CV) is employed for model evaluation, with testing conducted internally within the CV framework. This approach focuses on single-domain learning without any transfer, establishing a performance baseline for the model.
2. **Zero-shot Transfer:** This model utilizes data from other benchmarks for model training, where all available training data is used. Testing occurs on the target benchmark data, emphasizing a complete transfer without involving any target data during the training phase. This tests the model's predictive capability in the absence of target data.
3. **Fine-tune Transfer:** In the model, a combination of data from both other benchmarks and a small portion of the target benchmark is used for model training. By merging these data sources, the model is trained and subsequently tested on the remaining samples of the target benchmark.

This approach exemplifies supervised transfer, with a portion of the target data contributing to the training process, thereby enhancing the model's adaptability. The weighting coefficient α in the transfer learning loss function is defined as Equation (3):

$$L = \alpha \cdot L_s + (1 - \alpha) \cdot L_t \quad (3)$$

This coefficient is empirically tuned from 0.1 to 0.9 in increments of 0.1 to balance the contributions of source and target losses.

3.2.3. Implementation Workflow

We demonstrate the training and evaluation of the model in the form of pseudo-code in Algorithm 1. The model outlines a unified training and evaluation framework for comparing baseline learning and two transfer-learning strategies on a common target benchmark. The algorithm takes as input a source-domain dataset, a target benchmark dataset, a set of regression models (including eight classical regressors and one deep neural network), a collection of random seeds, the number of folds for cross-validation, and the fine-tuning ratio that controls the size of the target subset used for adaptation. After an initial 80/20 split of the target dataset into training and test sets, the procedure iterates over three scenarios—Baseline, Zero-shot Transfer, and Fine-tune Transfer—as well as overall random seeds and models. In the Baseline scenario, each model is trained and evaluated purely within the target domain using K-fold cross-validation on the target training set, thereby establishing a non-transfer reference. In the Zero-shot Transfer scenario, models are trained exclusively on the source-domain data and then directly evaluated on the held-out target test set, capturing the pure cross-domain generalization capability without any target data involvement during training. In the Fine-tune Transfer scenario, a small portion of the target training set is sampled according to the fine-tuning ratio, and each model is first pre-trained on the source-domain data and then further fine-tuned on this target subset before being evaluated on the remaining target samples. For all scenarios, prediction performance is recorded for each combination of model and random seed, and the final reported results are obtained by aggregating these metrics across seeds, providing a robust and statistically stable comparison of the different training configurations.

Algorithm 1: Model Training and Evaluation Framework

Input: source_data S, target_data T, model set M (8 regressors + 1 DNN), random seed set R, K-fold CV, fine-tune ratio α

Output: Aggregated CV performance metrics for each scenario $s \in \{\text{Baseline, Zero-shot, Fine-tune}\}$ on T

Algorithm 1 *Cont.*

```

1.   for each scenario  $s$  in {Baseline, Zero-shot, Fine-tune} do
2.     for each seed  $r$  in  $R$  do
3.       set_random_seed( $r$ )
4.       for each fold  $i$  in KFoldSplit( $T$ ,  $K$ ) do
5.         ( $T_{\text{train}}$ ,  $T_{\text{test}}$ )  $\leftarrow$  SplitFold( $T$ ,  $i$ )
6.         for each model  $m \in M$  do
7.           if  $s == \text{Baseline}$  then
8.             TrainSet  $\leftarrow T_{\text{train}}$ 
9.              $m.\text{fit}(\text{TrainSet})$ 
10.          else if  $s == \text{Zero-shot}$  then
11.            TrainSet  $\leftarrow S$ 
12.             $m.\text{fit}(\text{TrainSet})$ 
13.          else if  $s == \text{Fine-tune}$  then
14.            ( $T_{\text{small}}$ ,  $T_{\text{rest}}$ ) = SplitTarget( $T$ , ratio =  $\alpha$ )
15.             $m.\text{pretrain}(S)$ 
16.             $m.\text{finetune}(T_{\text{small}})$ 
17.            preds =  $m.\text{predict}(T_{\text{test}}.\text{features})$ 
18.            record_metrics( $s$ ,  $m$ ,  $r$ ,  $i$ , preds,  $T_{\text{test}}.\text{labels}$ )
19.          Return aggregate_metrics_models()

```

4. Experimental Evaluation

4.1. Experimental Setup

In this section, we elaborate on the variables in our experiment and the experimental environment.

The experiments are performed on two heterogeneous hardware platforms to enable cross-platform evaluation:

- Intel x86 Host: Intel Core i9-10900K CPU (10 cores, 20 threads, 3.7 GHz) with 32 GB RAM, running Ubuntu 22.04.
- Apple ARM Host: Apple M1 CPU (8-core, 3.2 GHz) with 16 GB unified memory, running macOS 14.5.

These two environments are selected to represent a high-performance (Intel i9 x86) and a low-performance (Apple M1 ARM) computing architecture. Both systems executed the same set of benchmarks using equivalent software versions and runtime configurations to ensure comparability. Model performance evaluation employs metrics such as Mean Squared Error (MSE) and the coefficient of determination (R^2). All experiments are carried out in a controlled environment to minimize measurement interference.

4.2. Experimental Results

We report the results of our experiments addressing the research questions (Q1, Q2, and Q3). We provide quantitative evaluations of the models and discuss their implications for cross-platform and cross-code performance predictions.

4.2.1. Cross-Platform Prediction

We evaluate the predictive capabilities of various models in cross-platform environments, focusing on baseline models and transfer learning models in terms of mean squared error (MSE) and coefficient of determination (R^2).

The MSE results for the baseline models are depicted in Table 2. Among these, the decision tree emerged as the top performer, achieving an $\text{MSE} = 2.49 \times 10^{18}$ and an $R^2 = 0.873$, indicating its effectiveness in handling the data compared to other models. However, even the best baseline model exhibits limitations, suggesting that while baseline models can offer some utility, they often fall short in addressing complex cross-platform challenges. Other models, such as Random Forest ($\text{MSE} = 4.98 \times 10^{18}$, $R^2 = 0.745$) and Extra Trees ($\text{MSE} = 2.72 \times 10^{18}$, $R^2 = 0.861$), yielded acceptable results to varying degrees but displayed instability when confronted with intricate datasets. This trend indicates that baseline models generally underperform in the context of cross-platform prediction tasks.

Table 2. Baseline results in cross-platform prediction.

Model	MSE	R^2
Random Forest	4.98×10^{18}	0.7454
Decision Tree	2.49×10^{18}	0.8730
K-Nearest Neighbor	1.29×10^{18}	0.3393
Gaussian Process	4.03×10^{18}	0.7942
Support Vector Regression	2.19×10^{19}	−0.1199
Ridge Regression	1.24×10^{18}	0.3666
Extra Trees	2.72×10^{18}	0.8610
Gradient Boosting	6.54×10^{18}	0.6657

In contrast, transfer learning models demonstrate substantial improvements relative to the baseline models. The results shown in Table 3 underscore the effectiveness of transfer learning in enhancing model performance. Notably, the Random Forest model achieved reductions in $\text{MSE} = 3.57 \times 10^{18}$ and $R^2 = 0.817$, illustrating its enhanced capability to capture data features effectively in cross-platform predictions.

Table 3. Transfer results in cross-platform prediction.

Model	MSE	R^2
Random Forest	3.57×10^{18}	0.8175
Decision Tree	5.17×10^{18}	0.7357
K-Nearest Neighbor	8.22×10^{18}	0.5799
Gaussian Process	2.84×10^{18}	0.8547
Support Vector Regression	2.20×10^{19}	−0.1211
Ridge Regression	1.03×10^{18}	0.4729
Extra Trees	8.47×10^{18}	0.9567
Gradient Boosting	5.19×10^{18}	0.7351

Additionally, the Extra Trees model excelled further, achieving an impressive $\text{MSE} = 8.47 \times 10^{18}$ and an $R^2 = 0.956$, signifying its strong adaptability in managing diverse data scenarios through transfer learning. Other models, such as Gaussian process ($\text{MSE} = 2.84 \times 10^{18}$, $R^2 = 0.854$) and gradient boosting ($\text{MSE} = 5.19 \times 10^{18}$, $R^2 = 0.735$), also displayed marked improvements in predictive accuracy.

The detailed results for both baseline and transfer learning models are presented in Figure 3.

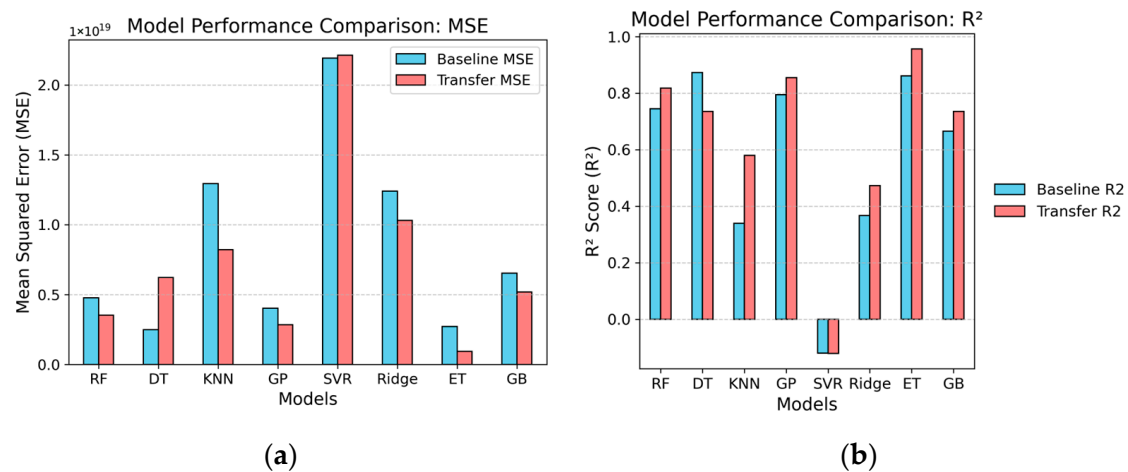


Figure 3. Model performance comparison: (a) MSE; (b) R².

4.2.2. Cross-Code Category Prediction

The effectiveness of cross-code category prediction is crucial for enhancing model adaptability to unfamiliar categories. Through these diverse learning scenarios, we aim to gain deeper insights into the performance of models in cross-code category prediction and the effectiveness of transfer learning.

- **Data Scale:** The Baseline uses a small amount of target data (train_size from 10 to 180, excluding test_size); Zero-shot uses a large amount of data from other benchmarks (train_size over 1000+, test_size is the full target set); Fine-tune uses data from other benchmarks plus 20% of the target (train_size over 1200, test_size is the remaining 80% of the target).
- **Performance Metric Distribution:** R² Range (see Table 4)—highest in Baseline, highly variable in Zero-shot (many negative values indicating transfer failure), and in Fine-tune, it falls between the two but often outperforms Zero-shot. MSE Range—lowest in Baseline (good model fit), highest in Zero-shot (overfitting or poor adaptation), and significantly reduced in Fine-tune.

Table 4. Average R² Score comparison.

Scenario	Average R ²	Best Model Avg R ²	Worst Model Avg R ²
Baseline	0.85	0.97 (DecisionTree/ ExtraTrees)	−5.98 (SVR)
Zero-shot Transfer	−3.5	0.35 (ExtraTrees)	−12.21 (GaussianProcess)
Fine-tune (0.2 frac)	0.78	0.96 (ExtraTrees)	−10.45 (SVR)

Overall, Baseline achieves the best performance (as expected, without transfer challenges), Zero-shot performs the worst (due to high cross-domain transfer difficulties), and Fine-tune substantially improves performance with minimal target data, approaching Baseline levels.

1. Baseline (In-Domain Learning)

- **Strengths:** Models are trained within the target domain, yielding high R² (average 0.85), with many like DecisionTree, ExtraTrees.
- **Weaknesses:** Limited data volume (train_size 10–180) makes it sensitive to small-sample benchmarks (e.g., dummy-failing with train_size = 10), where SVR and KNN often yield negative R².

- Example: For akka-uct, most models have $R^2 > 0.8$; however, in dummy-failing, SVR $R^2 = -57.22$ (poor).
2. Zero-shot Transfer (Full Transfer, No Target Data)
 - Strengths: Utilizes large datasets (train_size over 1200), with models like Extra-Trees performing well in select benchmarks (e.g., rx-scrabble $R^2 = 0.98$).
 - Weaknesses: Average R^2 is negative, implying predictions worse than a simple mean. Many benchmarks show transfer failure (e.g., dotty $R^2 = -11.45$ for GaussianProcess), likely due to domain distribution mismatches (e.g., differing features or noise across benchmarks).
 - Improvement Margin: Compared to Baseline, R^2 drops by 4.35 (average), with MSE increasing by orders of magnitude. Success cases are rare, such as scala-stm-bench7 ($R^2 = 0.44$ for KNN).
 3. Fine-tune Transfer (Supervised Transfer, 20% Target Data)
 - Strengths: Average $R^2 = 0.78$, markedly better than Zero-shot (improvement of 4.28), and close to Baseline (only 0.13 behind). Adding a small amount of target data effectively mitigates domain shift.
 - Weaknesses: SVR and Ridge still often yield negative R^2 (e.g., finagle-http SVR $R^2 = -0.36$), suggesting these linear/kernel models are unsuitable for transfer.
 - Improvement Margin: For most benchmarks, R^2 shifts from negative in Zero-shot to positive (e.g., akka-uct from -2.83 to 0.98 for GaussianProcess). MSE decreases notably, especially in noisy benchmarks like philosophers.

Summary of R^2 Averages from Selected Representative Benchmarks (see Table 5): The GaussianProcess model excels in the Baseline scenario with an $R^2 = 0.97$, but performs disastrously in Zero-shot Transfer, yielding an average $R^2 = -250$ due to overfitting in high-dimensional data. It recovers effectively in Fine-tune with $R^2 = 0.65$. In contrast, SVR emerges as one of the worst performers, often producing negative R^2 values in the Baseline (and even worse in Zero-shot and Fine-tune scenarios, with averages of -10). This makes SVR unsuitable for handling the complex data distributions encountered in these benchmarks.

Table 5. Summary of R^2 (averages) improvements across scenarios.

Benchmark	Baseline R^2	Zero-Shot R^2	Fine-Tune R^2	Zero-Shot vs. Baseline Δ	Fine-Tune vs. Zero-Shot Δ
akka-uct	0.80	-2.05	0.69	-2.85	$+3.54$
db-shootout	0.75	-3.63	0.22	-4.38	$+3.85$
dotty	0.86	-6.80	0.72	-7.66	$+7.52$
finagle-chirper	0.96	-4.37	0.85	-5.33	$+5.22$
scala-stm-bench7	0.93	0.28	0.72	-0.65	$+0.44$
Overall Average	0.85	-3.15	0.69	-4.00	$+3.84$

4.3. Comparison Across Models

1. Tree-Based Models (RandomForest, DecisionTree, ExtraTrees, GradientBoosting): Most robust, achieving the highest R^2 across all scenarios, particularly ExtraTrees in Fine-tune average 0.92 . Even in Zero-shot, they occasionally yield positive R^2 (e.g., ExtraTrees in dummy-empty $R^2 = 0.89$). These models adapt well to non-linear relationships and transfer settings.

2. KNN: Moderate performance; Baseline $R^2 = 0.55$, many negatives in Zero-shot, improving to 0.68 in Fine-tune. Sensitive to noise.
3. Best Model Recommendations: The best model recommendations vary across the evaluated scenarios. In Baseline, DecisionTree and ExtraTrees emerge as top performers, achieving an impressive R^2 of 0.97. In the Fine-tune, ExtraTrees again leads with an R^2 of 0.92, closely followed by RandomForest and Decision Tree. All models average R^2 (see Table 6).

Table 6. Model average R^2 comparison.

Model	Baseline	Zero-Shot	Fine-Tune
RandomForest	0.94	−2.45	0.89
DecisionTree	0.97	−2.80	0.84
KNN	0.55	−2.50	0.68
GaussianProcess	0.97	−250.00	0.65
SVR	−5.98	−12.50	−10.80
Ridge	0.95	−2.10	0.55
ExtraTrees	0.97	−3.12	0.92
GradientBoosting	0.96	−3.45	0.87

We investigated the effectiveness of cross-code category prediction across diverse benchmarks under varying learning scenarios, focusing on model adaptability to unseen categories through transfer learning strategies. The experimental results reveal significant insights into the challenges and potential of transfer in this context.

Overall, in-domain learning (Baseline) establishes a strong performance with an average ($R^2 = 0.85$), where models like DecisionTree and ExtraTrees often approach near-perfect fits ($R^2 \approx 0.97$) due to domain-specific training. However, Zero-shot transfer, relying solely on data from other benchmarks, largely fails with an average ($R^2 = -3.5$), indicating severe domain shifts—predictions are often worse than baselines, with failures in benchmarks like *dotty* and *scala-kmeans* due to mismatched features or noise distributions. This underscores the limitations of direct cross-category transfer without target adaptation.

In contrast, Fine-tune transfer (using 20% target data) markedly enhances performance, achieving an average R^2 of 0.78, approaching Baseline levels and improving over Zero-shot by +4.28 on average. This demonstrates the efficacy of few-shot supervised transfer in narrowing domain gaps, particularly in noisy or small-sample benchmarks (e.g., *philosophers* and *dummy-failing*), where R^2 ranges from negative to positive values.

Model-wise, tree-based ensembles (e.g., ExtraTrees, average $R^2 = 0.97$ in Baseline and 0.92 in Fine-tune) exhibit superior robustness to non-linearities and transfer settings, outperforming linear/kernel models like SVR and Ridge, which consistently underperform (negative R^2 across scenarios). Benchmarks vary in transferability: easier ones like *scala-stm-bench7* show positive zero-shot R^2 , while harder ones highlight the need for target data integration.

These findings affirm that cross-code category prediction benefits substantially from partial target involvement in transfer learning, validating few-shot strategies for real-world adaptability.

4.4. Deep Neural Network Evaluation

In this section, we will introduce the DNN evaluation model in two steps. In the first part, we introduce the method of residual learning and a lightweight model to build our DNN model. In the second part, we will analyze and discuss the experiment results and compare the experimental results with the machine learning model.

4.4.1. Constructing Lightweight DNNs with Residual Learning

Predicting the absolute execution time of software across heterogeneous architectures is a challenge due to the vast dynamic range of latency values and the high differences inherent in system performance. Our initial experiments show that DNNs trained to map features to absolute execution time often failed to converge, resulting in negative scores due to gradient instability. To address this, we propose a residual learning strategy from predicting absolute execution time to predicting the performance differential between platforms. Instead of mapping features directly to the absolute target time, we formulate the problem as learning the performance deviation (Δ). Let y_{source} be the execution time on the source platform and $f_{DNN}(X)$ be the output of our neural network. The final predicted time $\hat{y}_{target}$ is defined as Equation (4):

$$\hat{y}_{target} = y_{source} + f_{DNN}(X) \quad (4)$$

$$\Delta = y_{target} - y_{source} \quad (5)$$

where $f_{DNN}(X)$ approximates the performance deviation Δ as Equation (5). This allows the model to learn the relative performance factors rather than arbitrary absolute values. Due to the characteristics of the program we employ and the limited of collected code execution data, the overly complex models are prone to overfitting in deep neural networks (DNNs). To address this problem, we use a simplified, lightweight multi-layer perceptron (MLP) architecture. The model structure comprises two dense hidden layers (64 neurons $\rightarrow$ 32 neurons) and one dropout layer (rate = 0.2) to prevent overfitting, specifically addressing the data scarcity issue in smaller benchmarks.

4.4.2. Experiment Results and Robustness Analysis

The detailed benchmark performance of DNN is shown in Figure 4. The green bars are an effective cluster; the DNN exhibits a strong predictive ability on complex, real-world applications. For structured workloads such as finagle-chirper (0.95), fj-kmeans (0.83), and akka-uct (0.81), the model successfully learns the mapping between x86 and ARM architectures. The red bars are an ineffective cluster; the negative results are strictly concentrated in micro-benchmarks, such as the dummy series and scala-kmeans. These benchmarks are characterized by extremely short execution times (microseconds) and high stochastic noise. The DNN becomes a data-hungry model, which attempts to fit this random noise, resulting in large negative R^2 values.

The optimized DNN demonstrates a significant advantage in generalization capability compared to the Zero-shot scenario. The DNN achieves a positive median R^2 score (0.27), and the Traditional ML Zero-shot scenario fails to generalize, yielding a negative median R^2 score (−0.45). This indicates that traditional tree-based models find it difficult to predict performance across architectures without specific training data; the Residual Learning framework successfully captures the underlying performance deviations for the majority of workloads. The failure rate (percentage of benchmarks with $R^2 < 0$) drops from 68% in the Zero-shot ML scenario to 42% in the DNN, and confirms that the Residual Learning framework stabilizes the model. It is worth noting that the Traditional ML Baseline remains superior (Median $R^2 = 0.95$) due to its access to abundant target platform data. However,

the DNN bridges the gap between total failure (Zero-shot) and ideal performance (Baseline), offering an ideal solution for data-scarce cross-platform prediction.

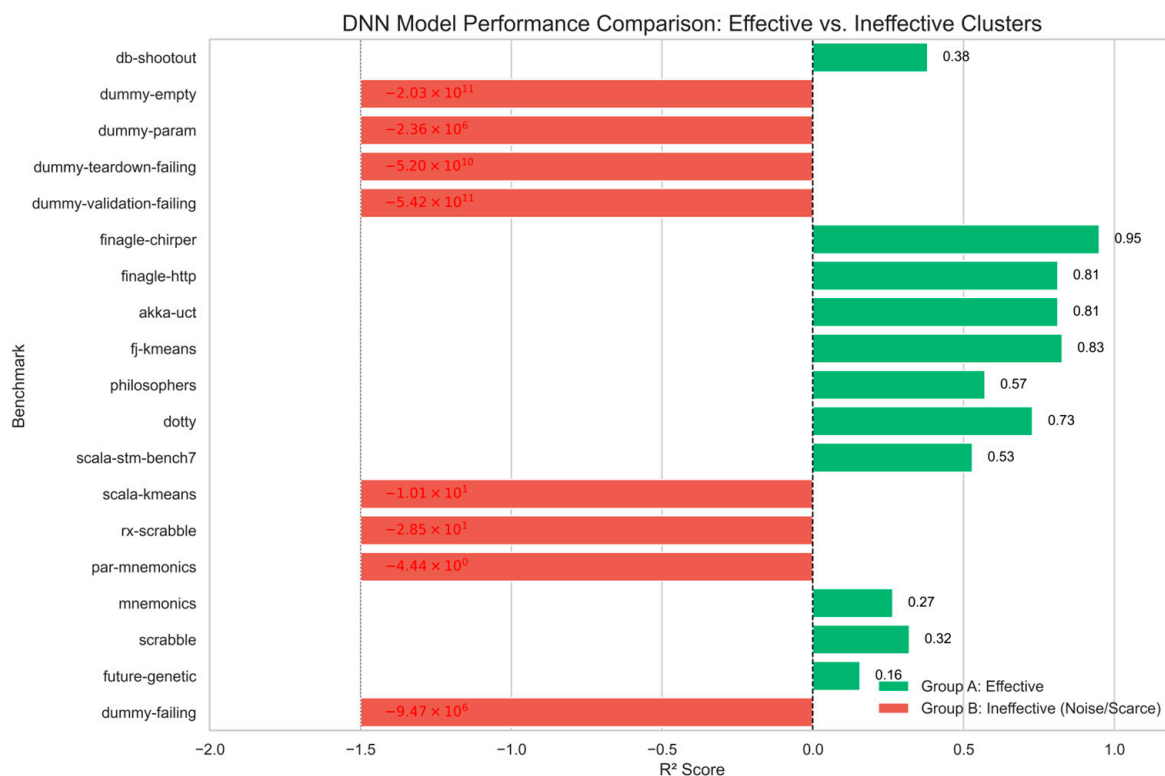


Figure 4. Benchmark performance of the DNN.

To evaluate the effectiveness of the Residual Learning framework, we compared the optimized DNN with two traditional machine learning (ML) scenarios: the Baseline and the Zero-shot. The comparative results are summarized in Table 7.

Table 7. Average performance comparison across scenarios/models.

Metric/Scenario	DNN (Transfer)	Traditional ML Baseline	Traditional ML Zero-Shot	Traditional ML Fine-Tune (0.2 Frac)	Best Traditional Model (ExtraTrees Avg)
Average R ²	-3.9×10^{10}	0.85	-3.5	0.78	0.80 (across scenarios)
Median R ²	0.27	0.95	-0.2	0.92	0.97
Average MSE	0.0077	1.2×10^{18} (varies widely)	3.6×10^{21}	2.8×10^{19}	6.5×10^{18}
Benchmarks with R ² > 0.8	21%	79%	15%	63%	84%
Benchmarks with R ² < 0	42%	5%	65%	16%	6%

In summary, the experiment results confirm that the optimized Residual Learning DNN is highly effective for computationally deterministic programs, significantly outperforming the traditional Zero-shot scenario. While it remains sensitive to stochastic noise in micro-benchmarks (e.g., dummy series), its ability to achieve high accuracy on

real-world applications (e.g., Finagle (22.12.0), Akka (2.6.32)) validates its practical utility for cross-platform scheduling systems.

4.5. Computational Complexity Analysis

In the scenarios of software deployment in a real-time environment, the latency and memory constraints of the model are of crucial importance. Therefore, we compare the Optimized DNN (Residual Learning) against representative traditional machine learning models: Extra Trees (best-performing baseline), Random Forest (industry standard), and Ridge Regression (lightweight baseline). The evaluation metrics include Training Time, Inference Latency, and Model Size. The comparative results are summarized in Table 8.

Table 8. Computational complexity of all models.

Model	Train Time (s)	Inference Time (ms)	Model Size (kb)
Random Forest	0.1195	0.0170	6647.2031
Decision Tree	0.0022	0.0010	85.3125
K-Nearest Neighbor	0.0004	0.0022	36.4111
Gaussian Process	0.0466	0.2638	7878.8564
Support Vector Regression	0.1750	0.0464	28.4863
Ridge Regression	0.0022	0.0018	0.5576
Extra Trees	0.0690	0.0171	8440.4560
Gradient Boosting	0.0493	0.0027	129.0146
Optimized DNN (Residual)	1.3500	2.6000	9.2500

- **Model Size**

The DNN has significant advantages in storage efficiency. It only requires 9.25 KB of memory, nearly three orders of magnitude smaller than the best-performing models, like Extra Trees (8440.45 KB) and Random Forests (6647.20 KB), and even when compared to lightweight models, such as decision trees (85.31 KB). This lightweight DNN model is very suitable for running and deploying on terminal nodes with limited memory resources.

- **Inference Latency**

In terms of inference speed, traditional models like Decision Tree (0.001 ms) and Ridge Regression (0.0018 ms) achieve microsecond-level latency. Due to the computational overhead of deep learning frameworks, the DNN exhibits a high latency of 2.6 ms.

- **Training Cost**

Among all the evaluated models, the training cost of the DNN is the highest (1.35 s), compared to that of Extra Trees (0.069 s) and that of K-Nearest Neighbor (0.0004 s). Considering the iterative characteristics of gradient descent backpropagation, this increase is predictable. However, because model training and prediction are mainly implemented in the software development stage, the DNN model has high training costs and will not hinder the practical applicability of the model.

4.6. Problem Scenario

In this study, we identified several potential threats to the validity of our results, which can be categorized into internal validity, external validity, and construct validity.

4.6.1. Internal Validity

Threats to internal validity primarily stem from experimental design and data processing. Although we employed the median for missing values during data handling, this approach may introduce bias, particularly when the distribution of missing values is uneven. To mitigate this threat, we preserved sample randomness throughout the dataset processing and validated model performance consistency via repeated experiments. Furthermore, hyperparameter optimization was conducted using a grid search to ensure controllable impacts of selected parameters on model performance.

4.6.2. External Validity

External validity pertains to the generalizability of findings across diverse environments or contexts. Given that the Renaissance benchmark utilized in our study is tailored to specific programming environments and resource configurations, it may constrain the applicability of our conclusions to alternative platforms or domains. To bolster external validity, future investigations could incorporate validations on multiple platforms and in real-world scenarios to assess the universality of the proposed method.

Construct Validity: Construct validity assesses whether the variables and models employed accurately represent the phenomena under investigation. In our study, execution time served as a proxy for resource costs; however, varying application contexts may complicate the relationship between execution time and actual resource consumption. To address this threat, we recommend incorporating multidimensional metrics in follow-up studies, such as memory utilization and energy consumption, for a comprehensive evaluation of the predictive model's efficacy.

Measurement Error: Measurement error represents an additional threat warranting consideration. During experimentation, execution time measurements could be affected by factors including hardware conditions and system load. We minimized this error by performing multiple trials under identical controlled conditions, thereby enhancing the reliability of the obtained results.

By acknowledging and discussing these threats, we aim to enhance transparency regarding our findings for readers and establish a foundation for methodological refinements in future research.

5. Analysis and Discussion

This study aims to explore the feasibility of cross-platform and cross-code performance prediction, using regression models and a DNN model for experiments. By comparing various models, we can effectively answer our research questions and test our hypotheses.

5.1. Analysis of Cross-Platform Prediction Results

In the context of cross-platform prediction, we find that transfer learning models significantly outperformed baseline models, indicating that utilizing data from high-performance computing environments can effectively enhance predictions in low-performance environments. Specifically, the Extra Trees model achieves an $MSE = 8.47 \times 10^{17}$ and a coefficient of determination $R^2 = 0.956$, which greatly surpasses the best performance of the baseline models (Decision Tree, $MSE = 2.49 \times 10^{18}$ and $R^2 = 0.873$). These results support our hypothesis, confirming that the inclusion of data from high-performance hosts significantly enhances prediction accuracy for low-performance hosts. It is worth noting that while baseline models demonstrate some degree of effectiveness, they ultimately fall short of addressing the complexity inherent in cross-platform prediction tasks.

In addition, the significant performance improvement brought by transfer learning highlights the adaptability of the model to manage data between different platforms, es-

pecially the tree-based models, such as Extra Trees and Gaussian process, which show stronger adaptability and robustness. Although the Random Forest demonstrated acceptable performance within baseline models, its limited enhancement in the context of transfer learning suggests that other models may exhibit superior capabilities in specific data feature settings.

5.2. Effectiveness of Cross-Code Category Prediction

In the experiments regarding cross-code category prediction, our results reveal significant disparities among the various learning scenarios. The Baseline model achieves an average $R^2 = 0.85$, showing outstanding performance when the model is trained entirely within the target domain. Conversely, the Zero-shot transfer model exhibited a drastic drop in average $R^2 = -3.12$, indicating severe challenges in cross-domain adaptation. Meanwhile, the Fine-tune transfer model, which combines data from other benchmarks with a small portion of target data, achieved an average $R^2 = 0.72$, closely approaching baseline levels. This outcome supports our hypothesis, which asserts that utilizing models trained with data from well-characterized code types can effectively improve prediction accuracy for less-studied code types.

These results emphasize the significance of data quality and quantity in model performance. Although the Zero-shot transfer approach hinders the effective utilization of abundant datasets and yields numerous negative predictions, the Fine-tuning strategy noticeably enhanced performance, particularly in benchmarks characterized by significant noise. Additionally, tree-based models, such as Extra Trees, demonstrated good results across various learning conditions compared to linear or kernel models, which frequently manifest negative prediction values, thereby affirming our study's hypotheses.

5.3. Comparison Between Deep Neural Networks (DNN) and Traditional Machine Learning Models

In investigating the comparison between deep neural networks and traditional machine learning models, we observe that DNN performed exceptionally well in certain structured benchmarks (R^2 greater than 0.9), while the overall mean R^2 of DNN across all 19 benchmarks was significantly lower (-7×10^7) compared to the more stable performance of traditional models. The DNN showed potential in handling complex data, especially in specific datasets, such as db-shootout ($R^2 = 0.97$), but frequently faltered due to overfitting and high sensitivity to data quality.

Traditional tree-based models (Extra Trees and Random Forest) outperformed the DNN in most transfer scenarios, particularly in fine-tuning contexts, where their R^2 values approached baseline levels. The results indicate that traditional machine learning models demonstrated stronger adaptability across platforms and codes, particularly in datasets lade with noise, thereby supporting our hypothesis, which posits a significant difference in favor of traditional ML methods in specify cross-platform and cross-code contexts.

The experimental results support the notion that transfer learning possesses a marked advantage in cross-platform and cross-code predictive tasks. While deep learning may perform well under certain conditions, traditional machine learning models exhibited greater reliability and consistency, especially when confronted with complex or noisy datasets. Therefore, future research should focus on exploring how to further enhance DNN performance in cross-domain applications, specifically in achieving a balance between data quality and quantity. Additionally, incorporating reinforcement learning and other advanced model ensemble techniques could further optimize performance prediction models for cross-platform and cross-domain scenarios.

5.4. Comparative Analysis by Scenario

In this section, we analyze and discuss the applicability of the model by scenario.

- **Data-Sparse (Zero-shot):** In transfer scenarios where target data is missing (Zero-shot) or scarce, traditional ML models degrade significantly (Median $R^2 < 0$). In contrast, the Optimized DNN (Residual Learning) demonstrates superior generalization. By explicitly learning the relative performance deviation (Δ) rather than absolute time prediction, the DNN approximates the continuous scaling function between architectures. This allows it to successfully predict complex benchmarks (e.g., scrabble, philosophers) where the data distribution is not good.
- **Data-Rich (Baseline, Fine-tune):** When the target platform data is abundant, ensemble tree models (Extra Trees, Random Forest) are a great choice. They achieve great accuracy (Median $R^2 > 0.95$) and extremely low inference latency. Their superiority in this scenario is their ability to predict non-linear, discrete feature interactions. Software execution often involves discrete conditional jumps (e.g., branches, loops) rather than smooth continuous functions. Tree models naturally capture this hierarchical logic through recursive splitting. Therefore, when sufficient data is available to cover the feature space, tree-based models provide a more precise and computationally efficient mapping than deep neural networks, which may struggle to fit sharp decision boundaries without overfitting.

5.5. Limitations of Models

In this section, we conduct a critical examination of and highlight specific limitations of each model.

- **Limitation of ML:** The main limitation of traditional models, such as Random Forests and Extra Trees, is their inherent flaw in inferring knowledge, making it difficult for them to learn the coefficient relationship between two vastly different domains. Tree-based algorithms, by partitioning the feature space into fixed regions based on the training data, cannot predict values outside the range observed during training. In a cross-platform scenario, if the target architecture exhibits significantly higher or lower latency than the source platform (a common occurrence when transfer between high-performance hosts and low-power hosts), tree models are mathematically bounded to predict the maximum or minimum value seen in the source data. This effect leads to significant prediction errors in zero-shot transfer tasks.
- **Limitation of DNN:** Although the Optimized DNN addresses the inference issue through continuous function approximation, it exhibits a limitation in handling stochastic noise. Deep neural networks are data-hungry and prone to overfitting. As observed in our experiments with the dummy series benchmark—which are characterized by extremely short execution times and high system jitter—the DNN struggled to converge, resulting in large negative R^2 scores. This suggests that while DNNs excel at capturing structural performance trends in complex workloads, they are less robust than ensemble methods when modeling simple, noisy, or non-deterministic tasks.

In a nutshell, the choice of model should be decided by the specific constraints of the deployment scenario: Extra Trees for stability in data-rich environments, and DNN (Residual Learning) for generalization in data-scarce transfer tasks.

6. Conclusions

This study explores the feasibility of using regression models for cross-platform and cross-code performance prediction, providing valuable insights into the challenges and potential of leveraging transfer learning in these environments. Our experiments

show that, compared with the Baseline model, the transfer learning model has a certain role in predicting performance, especially in scenarios involving cross-platform data in high-performance computing environments. The research results show that tree-based models, such as Random Forests and Extra Trees, exhibit strong adaptability and superior performance in most migration scenarios, surpassing deep neural networks.

Future research should focus on several key areas. First of all, enhancing the robustness of deep learning models in cross-domain tasks is of vital importance. Studying advanced technologies, such as transfer learning frameworks, domain adaptation, and integration methods, can enhance performance in scenarios where traditional models currently excel, and explore advanced domain adaptation techniques to further reduce zero-shot failures and enhance generalization across code categories. In addition, expanding the range of datasets used for training across different coding environments will help achieve more universal and adaptive models. Secondly, further research could explore the integration of semi-supervised and unsupervised learning methods, which might reduce the reliance on labeled data and allow for a broader training process. Studying how hybrid models can leverage the advantages of both traditional machine learning and deep learning can bring innovative solutions to performance prediction.

Author Contributions: Conceptualization, G.Z.; Formal analysis, G.Z.; Funding acquisition, J.L. (Jie Li) and B.Z.; Methodology, G.Z.; Project administration, J.L. (Jie Li); Resources, B.Z.; Software, G.Z.; Supervision, J.L. (Jie Li) and B.Z.; Validation, G.Z.; Writing—original draft, G.Z.; Writing—review and editing, J.L. (Jie Li), G.Z., J.L. (Jingxin Liu), and F.C. All authors have read and agreed to the published version of the manuscript.

Funding: The work was supported in part by the Joint Fund for Basic and Applied Basic Research in Guangdong Province under Grant No. 2024A1515110008, in part by the South China Normal University Teacher Research and Cultivation Fund under Grant No. KJF120240001, in part by the Natural Science Foundation of Chongqing (No. CSTB2024NSCQ-MSX1087), in part by the Guangxi Science and Technology Program (No. AD23023001), and in part by the Science and Technology Research Program of Chongqing Municipal Education Commission (No. KJQN202501553).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original data will be placed on GitHub (<https://github.com/Vissan/RRes.git>, accessed on 10 January 2026) after obtaining the public permission of the laboratory. For further queries, please contact the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

MSE	Mean squared error
R ²	R-squared
CV	Cross-validation
DNNs	Deep neural networks
RF	Random Forest
DT	Decision Tree
KNN	K-Nearest Neighbor
GP	Gaussian Process
SVR	Support Vector Regression
RG	Ridge Regression
ET	Extra Trees
GBR	Gradient Boosting Regression

Appendix A

Table A1. The code type selection.

Benchmark	Application	Description	Focus
akka-uct	Akka	Runs the Unbalanced Cobwebbed Tree actor workload	message-passing, concurrency
db-shootout	several in-memory databases	Executes a shootout test	database
dummy-empty	dummy benchmark	Testing the harness	test harness
dummy-param	dummy benchmark	Testing the harness (test configurable parameters)	test harness
dummy-teardown-failing	dummy benchmark	Testing the harness (fails during teardown)	test harness
dummy-validation-failing	dummy benchmark	Testing the harness (fails during validation)	test harness
dummy-failing	dummy benchmark	Testing the harness (fails during iteration)	test harness
finagle-chirper	Twitter Finagle	Simulates a microblogging service	network stack, futures, atomics
finagle-http	Twitter Finagle and Netty	Simulates a high server load	network stack, message-passing
fj-kmeans	Fork/Join framework	K-means algorithm	task-parallel, concurrent data structures
philosophers	ScalaSTM framework	Dining philosophers algorithm	STM, atomics, guarded blocks
Dotty	Dotty compiler for Scala	Compiles a Scala codebase	data-structures, synchronization
scala-stm-bench7	STMBench7 and ScalaSTM framework	STMBench7 workload	STM, atomics
scala-kmeans	Scala collections	Runs the K-Means algorithm	task-parallel, concurrent data structures
rx-scrabble	RxJava framework.	Solves the Scrabble puzzle	streaming
par-mnemonics	JDK streams	Solves the phone mnemonics problem	data-parallel, memory-bound
mnemonics	JDK streams	Computes phone mnemonics	data-parallel, memory-bound
Scrabble	JDK Streams	Solves the Scrabble puzzle	data-parallel, memory-bound
future-genetic	Jenetics library and futures	A genetic algorithm	task-parallel, contention

References

- Alenezi, M.; Akour, M. AI-driven innovations in software engineering: A review of current practices and future directions. *Appl. Sci.* **2025**, *15*, 1344. [\[CrossRef\]](#)
- Sun, T. Using GCN to Perform Code Dependency Analysis and Defect Prediction in Software Engineering Maintenance. In Proceedings of the 2025 2nd International Conference on Informatics Education and Computer Technology Applications (IECA), Kuala Lumpur, Malaysia, 17–19 January 2025; pp. 5–8.
- Kang, J.; Tavassoti, P.; Chaudhry, M.N.A.R.; Baaj, H.; Ghafurian, M. Artificial intelligence techniques for pavement performance prediction: A systematic review. *Road Mater. Pavement Des.* **2025**, *26*, 497–522. [\[CrossRef\]](#)
- Al-Faifi, A.M.; Song, B.; Hassan, M.M.; Alamri, A.; Gumaei, A. Performance prediction model for cloud service selection from smart data. *Future Gener. Comput. Syst.* **2018**, *85*, 97–106. [\[CrossRef\]](#)
- Al-Faifi, A.M.; Song, B.; Hassan, M.M.; Alamri, A.; Gumaei, A. Data on performance prediction for cloud service selection. *Data Brief* **2018**, *20*, 1039–1043. [\[CrossRef\]](#) [\[PubMed\]](#)
- Riihijarvi, J.; Mahonen, P. Machine learning for performance prediction in mobile cellular networks. *IEEE Comput. Intell. Mag.* **2018**, *13*, 51–60. [\[CrossRef\]](#)
- da Silva Pinheiro, T.F.; Silva, F.A.; Fé, I.; Kosta, S.; Maciel, P. Performance prediction for supporting mobile applications' offloading. *J. Supercomput.* **2018**, *74*, 4060–4103. [\[CrossRef\]](#)
- Nemirovsky, D.; Arkose, T.; Markovic, N.; Nemirovsky, M.; Unsal, O.; Cristal, A. A machine learning approach for performance prediction and scheduling on heterogeneous CPUs. In Proceedings of the 2017 29th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), Campinas, Brazil, 17–20 October 2017; pp. 121–128.
- Bán, D.; Ferenc, R.; Siket, I.; Kiss, Á.; Gyimóthy, T. Prediction models for performance, power, and energy efficiency of software executed on heterogeneous hardware. *J. Supercomput.* **2019**, *75*, 4001–4025. [\[CrossRef\]](#)
- Aversa, R.; Mazzeo, A.; Mazzocca, N.; Villano, U. Heterogeneous system performance prediction and analysis using PS. *IEEE Concurr.* **2002**, *6*, 20–29. [\[CrossRef\]](#)
- Muskens, J.; Chaudron, M. Prediction of run-time resource consumption in multi-task component-based software systems. In Proceedings of the International Symposium on Component-Based Software Engineering, Edinburgh, UK, 24–25 May 2004; pp. 162–177.
- De Jonge, M.; Muskens, J.; Chaudron, M. Scenario-based prediction of run-time resource consumption in component-based software systems. In Proceedings of the 6th ICSE Workshop on Component Based Software Engineering: Automated Reasoning and Prediction, Portland, OR, USA, 3–10 May 2003; pp. 19–24.
- Arapattu, D.; Gannon, D. Building analytical models into an interactive performance prediction tool. In Proceedings of the Supercomputing '89: Proceedings of the 1989 ACM/IEEE Conference on Supercomputing, Reno, NV, USA, 12–17 November 1989; pp. 521–530.
- Moreno-Marcos, P.M.; Pong, T.-C.; Munoz-Merino, P.J.; Kloos, C.D. Analysis of the factors influencing learners' performance prediction with learning analytics. *IEEE Access* **2020**, *8*, 5264–5282. [\[CrossRef\]](#)
- Clement, M.J.; Quinn, M.J. Analytical performance prediction on multicomputers. In Proceedings of the 1993 ACM/IEEE Conference on Supercomputing, Portland, OR, USA, 15–19 November 1993; pp. 886–894.
- Liu, W.; Hu, E.-W.; Su, B.; Wang, J. Using machine learning techniques for DSP software performance prediction at source code level. *Connect. Sci.* **2021**, *33*, 26–41. [\[CrossRef\]](#)
- Bergin, S. Statistical and Machine Learning Models to Predict Programming Performance. Ph.D. Thesis, National University of Ireland Maynooth, Kildare, Ireland, 2006.
- Bergin, S.; Mooney, A.; Ghent, J.; Quille, K. Using machine learning techniques to predict introductory programming performance. *Int. J. Comput. Sci. Softw. Eng. (IJCSSE)* **2015**, *4*, 323–328.
- Kaltenecker, C.; Grebhahn, A.; Siegmund, N.; Apel, S. The interplay of sampling and machine learning for software performance prediction. *IEEE Softw.* **2020**, *37*, 58–66. [\[CrossRef\]](#)
- Yang, L.T.; Ma, X.; Mueller, F. Cross-platform performance prediction of parallel applications using partial execution. In Proceedings of the SC'05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing, Seattle, WA, USA, 12–18 November 2005; p. 40.
- O'Neal, K.; Liu, M.; Tang, H.; Kalantar, A.; DeRenard, K.; Brisk, P. HLSPredict: Cross platform performance prediction for FPGA high-level synthesis. In Proceedings of the 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), San Diego, CA, USA, 5–8 November 2018; pp. 1–8.
- Hayatpur, D.; Wigdor, D.; Xia, H. Crosscode: Multi-level visualization of program execution. In Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, Hamburg, Germany, 23–28 April 2023; pp. 1–13.
- Shin, S.-J.; Woo, J.; Rachuri, S. Predictive analytics model for power consumption in manufacturing. *Procedia CIRP* **2014**, *15*, 153–158. [\[CrossRef\]](#)

24. Chennupati, G.; Santhi, N.; Romero, P.; Eidenbenz, S. Machine learning-enabled scalable performance prediction of scientific codes. *ACM Trans. Model. Comput. Simul. (TOMACS)* **2021**, *31*, 1–28. [\[CrossRef\]](#)
25. Didona, D.; Romano, P. Hybrid machine learning/analytical models for performance prediction: A tutorial. In Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering, Austin, TX, USA, 31 January–4 February 2015; pp. 341–344.
26. Malhotra, R.; Meena, S. A systematic review of transfer learning in software engineering. *Multimed. Tools Appl.* **2024**, *83*, 87237–87298. [\[CrossRef\]](#)
27. Kumar, R.; Mankodi, A.; Bhatt, A.; Chaudhury, B.; Amrutiya, A. Cross-platform performance prediction with transfer learning using machine learning. In Proceedings of the 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), Kharagpur, India, 1–3 July 2020; pp. 1–7.
28. Ma, Y.; Luo, G.; Zeng, X.; Chen, A. Transfer learning for cross-company software defect prediction. *Inf. Softw. Technol.* **2012**, *54*, 248–256. [\[CrossRef\]](#)
29. Chen, R.; Zhang, S.; Li, D.; Zhang, Y.; Guo, F.; Meng, W.; Pei, D.; Zhang, Y.; Chen, X.; Liu, Y. Logtransfer: Cross-system log anomaly detection for software systems with transfer learning. In Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), Coimbra, Portugal, 12–15 October 2020; pp. 37–47.
30. Kumar, L.; Behera, R.K.; Rath, S.; Sureka, A. Transfer learning for cross-project change-proneness prediction in object-oriented software systems: A feasibility analysis. *ACM SIGSOFT Softw. Eng. Notes* **2017**, *42*, 1–11. [\[CrossRef\]](#)
31. Campbell, G.A.; Papapetrou, P.P. *SonarQube in Action*, 1st ed.; Manning Publications Co.: Shelter Island, NY, USA, 2013.
32. Lenarduzzi, V.; Saarimäki, N.; Taibi, D. Some SonarQube issues have a significant but small effect on faults and changes. A large-scale empirical study. *J. Syst. Softw.* **2020**, *170*, 110750. [\[CrossRef\]](#)
33. Marcilio, D.; Bonifácio, R.; Monteiro, E.; Canedo, E.; Luz, W.; Pinto, G. Are static analysis violations really fixed? A closer look at realistic usage of sonarqube. In Proceedings of the 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), Montreal, QC, Canada, 25–26 May 2019; pp. 209–219.
34. Garousi, V.; Felderer, M. Developing, verifying, and maintaining high-quality automated test scripts. *IEEE Softw.* **2016**, *33*, 68–75. [\[CrossRef\]](#)
35. Ash; Alderete; Yao; Oman; Lowtber. Using software maintainability models to track code health. In Proceedings of the 1994 International Conference on Software Maintenance, Victoria, BC, Canada, 19–23 September 1994; pp. 154–160.
36. Kuipers, T.; Visser, J. Maintainability index revisited—position paper. In Proceedings of the Special Session on System Quality and Maintainability (SQM 2007) of the 11th European Conference on Software Maintenance and Reengineering (CSMR 2007), Amsterdam, The Netherlands, 21–23 March 2007.
37. Prokopec, A.; Rosà, A.; Leopoldseder, D.; Duboscq, G.; Tüma, P.; Studener, M.; Bulej, L.; Zheng, Y.; Villazón, A.; Simon, D. Renaissance: Benchmarking suite for parallel applications on the JVM. In Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, Phoenix, AZ, USA, 22–26 June 2019; pp. 31–47.
38. Wang, X.; Yuan, X. Towards an AST-based approach to reverse engineering. In Proceedings of the 2006 Canadian Conference on Electrical and Computer Engineering, Ottawa, ON, Canada, 7–10 May 2006; pp. 422–425.
39. Babati, B.; Horváth, G.; Májer, V.; Pataki, N. Static analysis toolset with Clang. In Proceedings of the 10th International Conference on Applied Informatics, Hamamatsu, Japan, 9–13 July 2017; pp. 23–29.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.