# Predicting New Workload or CPU Performance by Analyzing Public Datasets

YU WANG, Harvard University
VICTOR LEE, Intel Corporation
GU-YEON WEI and DAVID BROOKS, Harvard University

The marketplace for general-purpose microprocessors offers hundreds of functionally similar models, differing by traits like frequency, core count, cache size, memory bandwidth, and power consumption. Their performance depends not only on microarchitecture, but also on the nature of the workloads being executed. Given a set of intended workloads, the consumer needs both performance and price information to make rational buying decisions. Many benchmark suites have been developed to measure processor performance, and their results for large collections of CPUs are often publicly available. However, repositories of benchmark results are not always helpful when consumers need performance data for *new processors* or *new workloads*. Moreover, the aggregate scores for benchmark suites designed to cover a broad spectrum of workload types can be misleading. To address these problems, we have developed a deep neural network (DNN) model, and we have used it to learn the relationship between the specifications of Intel CPUs and their performance on the SPEC CPU2006 and Geekbench 3 benchmark suites. We show that we can generate useful predictions for *new processors* and *new workloads*. We also cross-predict the two benchmark suites and compare their performance scores. The results quantify the self-similarity of these suites for the first time in the literature. This work should discourage consumers from basing purchasing decisions exclusively on Geekbench 3, and it should encourage academics to evaluate research using more diverse workloads than the SPEC CPU suites alone.

CCS Concepts: • **Computing methodologies** → *Machine learning*; *Machine learning approaches*; *Neural networks*;

Additional Key Words and Phrases: Performance prediction, performance comparison, benchmarking, data mining

**ACM Reference format:**
Yu Wang, Victor Lee, Gu-Yeon Wei, and David Brooks. 2019. Predicting New Workload or CPU Performance by Analyzing Public Datasets. *ACM Trans. Archit. Code Optim.* 15, 4, Article 53 (January 2019), 21 pages.
https://doi.org/10.1145/3284127

Part of this work was supported as an internship project at the Intel Parallel Computing Lab under the supervision of Victor Lee. This work was supported in part by the National Science Foundation (NSF) under award number 1533737.
Authors' addresses: Y. Wang, G.-Y. Wei, and D. Brooks, School of Engineering and Applied Sciences, Harvard University, Cambridge, MA, US; emails: {ywang03, dbrooks, gywei}@g.harvard.edu; V. Lee, Intel Corporation, Santa Clara, CA, US; email: victor.w.lee@intel.com.

1544-3566/2019/01-ART53
https://doi.org/10.1145/3284127

## 1 INTRODUCTION

Computer hardware evolves rapidly. In 2015 alone, Intel released over 200 CPU SKUs.[1] The SKUs have different characteristics like frequency, cache size, memory bandwidth, and core count. Better configurations usually cost more. The performance of a microprocessor is not solely a function of its microarchitecture; it depends critically on the nature of the workloads running on it. Thus, both CPU microarchitecture and workloads need to be taken into account when quantifying actual CPU performance. Consumers can then estimate whether investing in a costly configuration helps to meet their particular goals.

To help study processor performance, a variety of benchmark suites have been developed. For the more popular suites, such as Geekbench, SPEC CPU, and Passmark, sizable repositories of performance data have been collected, allowing the public to compare the behaviors of known configurations on standard workloads. However, there are some issues in using these repositories. The first is that consumers need to wait for comprehensive benchmark results of *new processors* to be contributed to the public repositories. The second issue is that for *new workloads*, consumers have to test a large number of possible configurations to find the most effective hardware. Daunted by this challenge, some consumers rely blindly on aggregate CPU scores in benchmark repositories.

To resolve these issues, we use statistical methods and machine learning techniques to analyze data from the SPEC CPU2006 and Geekbench 3 repositories.[2] SPEC CPU2006 is widely reported in academic studies. Geekbench 3 is used by many consumers to make purchasing decisions. We use deep neural networks (DNNs) to predict the performance of Intel CPUs, and we compare the DNN prediction with that by linear regression (LR). The DNN predictive model learns interactions between processor specifications for different workloads, sharpening the usefulness of benchmark data repositories.

This article makes the following contributions:

- We use DNN to predict the performance of SPEC and Geekbench workloads on new CPU SKUs. (The mean error is 5% for SPEC and 11% for Geekbench.) We show that DNN is more accurate than traditional LR.
- With the same accuracy, we show that the performance of new workloads on just 10 SKUs can predict their performance on other SKUs.
- For the first time in the literature, we quantify the self-similarity of these widely used suites, by using DNN to cross-predict the results between the suites (with 25.9% and 14.9% mean error when predicting SPEC and Geekbench, respectively), and by comparing their CPU rankings (which are inconsistent, with footrule distance as high as 0.59).

The rest of the article is organized as follows. Section 2 gives our analysis of Intel processors in terms of SKU specifications and SPEC performance. Section 3 describes our methodology. Section 4 summarizes our three case studies. Section 5 compares our prediction accuracy on those case studies. Section 6 compares the SKU rankings in the three cases. Section 7 summarizes related work and compares it with our own. Section 8 presents our conclusions.

## 2 INTEL PROCESSOR STATISTICS

In this section, we present some statistical facts about Intel processors, SPEC performance for the Intel SKUs, and the SKUs' microarchitectures and types in our SPEC and Geekbench datasets. We show the large performance variations and a rich variety of SKUs in our datasets.

[1]Stock Keeping Unit. In this article, a SKU is a CPU with a distinct processor number.
[2]The datasets in this article are available at https://github.com/Emma926/cpu_selection.git.

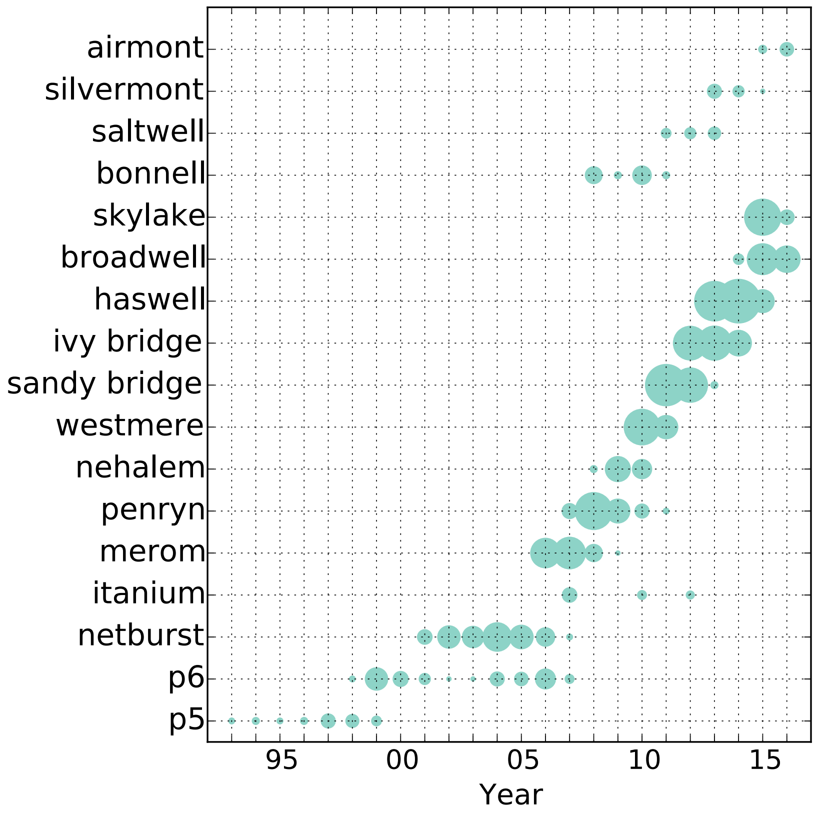

Fig. 1. The number of SKUs released by Intel for 17 microarchitectures between 1993 and 2016. Bubble size indicates the number of SKUs. Architectures include x86, Itanium, and Atom. (Data from http://ark.intel.com.)

We construct our Intel processor specification dataset from http://ark.intel.com, where Intel publishes the specifications of all its processors. We collect specifications including microarchitecture, launch year, number of cores, base frequency, cache size, power, and memory type. Figure 1 shows the release years of Intel microarchitectures. The size of a bubble corresponds to the number of SKUs released in the corresponding year. The figure contains 17 microarchitectures that Intel has released since 1993.

Intel has a "Tick-Tock" model for microarchitecture code names. A tick represents a shrinking of the technology and a tock represents a new microarchitecture. In Figure 1, for example, Westmere and Sandy Bridge have 32nm feature size, and Sandy Bridge and Ivy Bridge have the same microarchitecture. Every year, there is expected to be a tick or tock. Starting in 2014, however, Intel decided to add a "refresh" cycle, to update the current microarchitecture. That is why for 2014 in Figure 1 there are only a few Broadwell SKUs but more Haswell SKUs. After 2016, Intel switched their model to a three element cycle known as "Process-Architecture-Optimization."

In this article, we use relative runtime as a measure of performance. We take a Sandy Bridge processor (E3-1230) as the reference machine. We choose E3-1230 because it is common across datasets. The relative runtime of a workload is defined as

$$relative\ runtime = \frac{runtime - runtime_{ref}}{runtime_{ref}}, \tag{1}$$

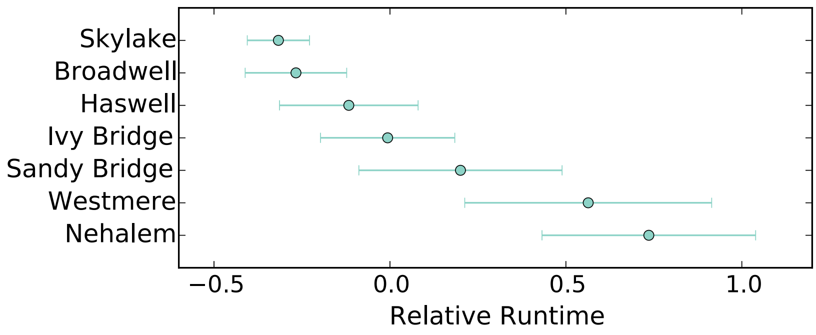

Fig. 2. Means and standard deviations of SPEC relative runtimes. Each runtime is the average over all SPEC workloads of performance for a given SKU. The lengths of the bars show that SKUs with the same microarchitecture have very diverse performance.

where $runtime_{ref}$ is the runtime of the corresponding workload on the reference machine. For brevity, this article will consistently use "performance" to indicate relative runtime. Thus, the scaled runtime of E3-1230 is 0. In Figure 2, the E3-1230 performance is slightly below the average of all Sandy Bridge SKUs, so the line is centered on a value greater than 0. Since all results are presented relative to the same reference processor, the results are unaffected by the particular choice of the reference.

To show the diverse performance of different SKUs, we take the average SPEC performance on SKUs, and we gather the SKUs based on their microarchitecture code names. In Figure 2, the dots show the mean relative runtime of the SKUs with the same microarchitecture code name, and the lengths of the bars show the standard deviations. Longer bars indicate the performance of the SKUs is more diverse. The SKUs have the same microarchitecture but different frequency, cache size, memory size, and so on. These lead to variations in each bar, and *they contribute as much to performance variation as the microarchitectures themselves.* The bars overlap horizontally. That means when selecting SKUs to optimize performance, there can be many choices with different microarchitectures but similar performance.

The SKUs in the public repositories are random, depending on users' submissions. SPEC and Geekbench have their own setup and compilation instructions. The setup of each data point is included in the repositories. We summarize the SKUs in the SPEC and Geekbench datasets in Figure 3. Our work focuses on performance prediction on the SKUs currently on the market, thus we discard SKUs older than Sandy Bridge. We first do preprocessing to aggregate duplicate data. Before explaining preprocessing, we first define a CPU configuration as a SKU running at a certain frequency with a certain memory size. In this article, we say that a configuration (SKU, frequency, memory size) determines a workload's performance. Other factors such as compiler and OS affect performance as well, but we do not consider them in this work. In the repositories, multiple result submissions may have the same configuration. We average the data points that have the same workload and configuration.

Figure 3 shows the microarchitectures and types of the SKUs in our SPEC and Geekbench datasets. There are 352 SKUs in total for SPEC, and 119 SKUs for Geekbench. In both datasets, we have more SKUs of Sandy Bridge, Ivy Bridge, and Haswell than Broadwell and Skylake. Figure 3 shows that we have a fairly rich collection of SKUs.

Users can customize the memory size with a chosen SKU. We refer to a SKU with a certain memory size as a configuration. In total, we have 639 configurations of 352 SKUs for the SPEC workloads. To help visualize the performance of SPEC workloads on 639 configurations, we first

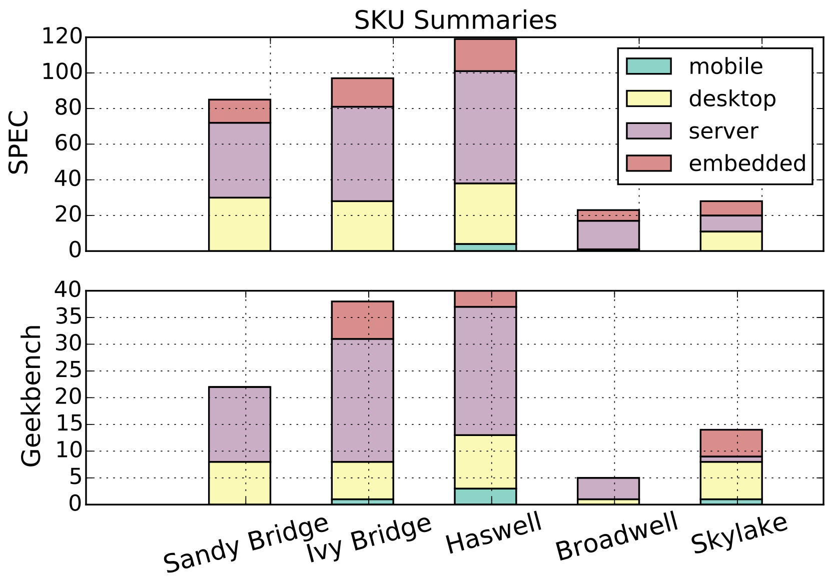

Fig. 3. SKU breakdown in the SPEC (top) and Geekbench (bottom) datasets. Total number of SKUs is 352 for SPEC and 119 for Geekbench.

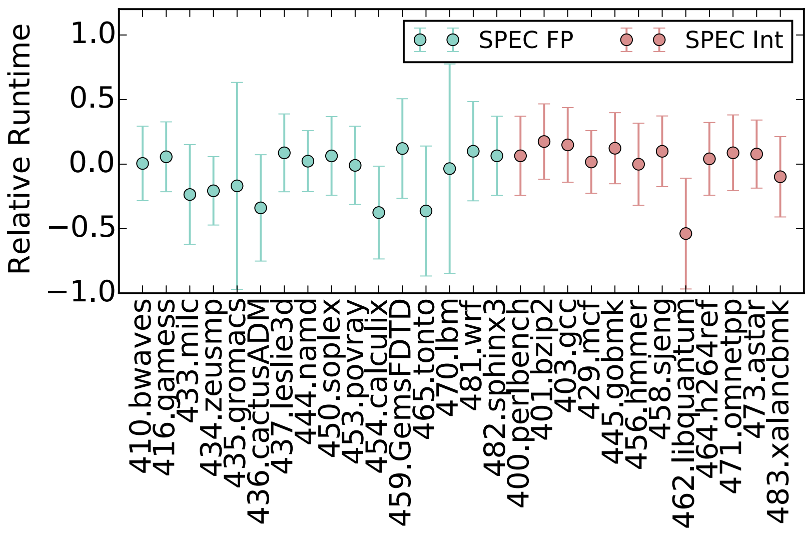

Fig. 4. Means and standard deviations of SPEC workloads' relative runtimes on the SKUs in our dataset. SPECfp is more diverse than SPECint. 462.libquantum is more similar to SPECfp.

show, in Figure 4, the means and standard deviations of the relative runtimes of the workloads. The blue bars are SPECfp; the red bars, SPECint. SPECint workloads are very similar to each other, except for 462.libquantum. SPECfp workloads have more diverse means and standard deviations. That indicates FP workloads are more diverse in terms of performance on the 639 configurations. The performance of 462.libquantum is more similar to SPECfp. If clustering the workloads, we would like to see that they form two clusters, with SPECint and some SPECfp workloads in one

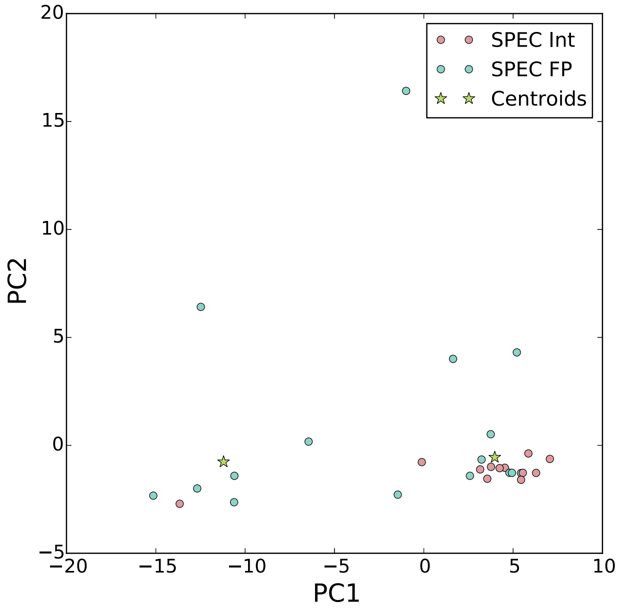

Fig. 5. Principal component analysis of SPEC workloads' performance scaling on 639 configurations of 352 SKUs. The green stars are centroids of two clusters identified by *k*-means. SPECfp workloads are more spread out than SPECint. 462.libquantum (red dot at the bottom left) is clustered with SPECfp benchmarks. The outlier near the top is 481.wrf.

cluster and the other FP workloads, which have quite different means and standard deviations in Figure 4, in the other cluster.

To visualize them in a two-dimensional space, we construct a matrix with 28 rows (for the 28 SPEC workloads) and 639 columns. Every matrix entry is the relative runtime of the workload on the corresponding SKU configuration. We then do a principal component analysis (PCA) to visualize the matrix. Figure 5 shows the PCA space. We only show two principal components (PCs) because the first and second PCs account for 82% of the variance.

The closer two workloads are, the more similar their performance scaling across the 639 configurations. Blue dots are SPECfp and red are SPECint. SPECfp is more spread out than SPECint. That means SPECfp workloads are more sensitive to SKU configuration differences, such as microarchitecture, frequency, and memory size. It is also an indication that it is harder for current microarchitectures to extract performance from SPECint than FP. SPECint workloads have less instruction level parallelism and more data dependencies. This is also observed by Campanoni et al. [2].

The points at the bottom of Figure 5 form two clusters, and the point at the top (481.wrf) is clearly an outlier. We want to find the centroids of the two clusters while leaving 481.wrf alone. Running a *k*-means algorithm with three clusters leaves 481.wrf in a cluster by itself. We plot the two centroids of the other two clusters using stars. 462.libquantum (the red dot at the bottom left) is clustered with SPECfp. The centroids will be used to explain our results in Section 5.2.

Although Figures 4 and 5 lead to consistent conclusions, they characterize the performance space differently. Figure 4 focuses on the means and standard deviations of the performance data, while PCA in Figure 5 emphasizes different performance on different SKU configurations. This can be explained with an example using two workloads (A, B) and three SKU configurations. Workload A's relative runtimes are 0.5, 0.7, −0.4, respectively. On the same SKU configurations, workload B's

Table 1. Data in This Article

| Data | Description | Data Type |
|---|---|---|
| Workload | The names of the workloads | One hot encode |
| Microarchitecture Code Name | Intel code names | Integer encode |
| Type | Server, desktop, mobile, or embedded | One hot encode |
| L3 Cache Size | Last level cache size | Numerical |
| Instruction Set Extensions | SSE or AVX | Integer encode |
| Memory Type | DDR,DDR2,DDR3,DDR4 | Integer encode |
| Memory Channels | Number of memory channels | Numerical |
| Max Memory Bandwidth | Max memory bandwidth | Numerical |
| ECC Memory Support | Whether ECC memory is supported | Binary |
| Base Frequency | Nominal frequency | Numerical |
| Turbo Frequency | Turbo frequency | Numerical |
| Turbo Boost Technology | Whether Turbo Boost is supported | Binary |
| Cores | Number of cores | Numerical |
| Threads | Number of threads | Numerical |
| Hyper-Threading | Whether hyper-threading is supported | Binary |
| TDP | Thermal design power | Numerical |
| Year | Year of release | Numerical |
| Frequency | The dynamic frequency | Numerical |
| Memory Size | The size of off-chip memory | Numerical |
| Runtime | The runtimes of the workloads | Numerical |

The data include CPU specifications (from the Intel processor dataset) and dynamic workload features (from SPEC and Geekbench datasets), and what we predict (runtime).

relative runtimes are 0.7, −0.4, and 0.5. The two workloads could look exactly the same in Figure 4, but PCA would show their difference.

## 3 METHODOLOGY

The previous section shows a large variation in performance. Thus, a predictive model can be helpful. This section shows the method we use to build a predictive model. We present our features, data preprocessing, model selection, and metrics (baselines) for evaluating the results.

A naive way to make a prediction is to use a previous generation to predict its successor. To do this, one needs the exact configuration of interest from previous generation, because sometimes other features are more important than microarchitecture generations, shown in the bars of Figure 2. For the same reason, using other SKUs in one generation to predict a new SKU in the same generation is difficult. We have tried that, and our current predictive models achieve better accuracy. We eventually decide to use machine learning to learn the relations between CPU specifications and performance, to avoid the missing data problem of the naive methods.

### 3.1 Features

Table 1 summarizes the data in this article. The data include features fed to the predictive model, and the value we predict, the runtime. The features include CPU specifications from the Intel CPU dataset and dynamic workload features from the SPEC and Geekbench datasets.

**CPU Specifications:** The microarchitecture code names include microarchitecture changes and technology scaling. Within one microarchitecture generation, the SKUs are different in their frequencies, last level cache sizes, TDP, and number of cores. We also add the release year as a feature. The type of the SKU is also an important feature. Intel categorizes the SKUs as mobile, desktop, server, and embedded. L2 and L1 caches per core do not change much across SKUs. Thus, we only include last level cache sizes.

For discrete data, we do integer encoding and one hot encoding. Integer encoding means mapping discrete values to integers. Each feature needs only one entry. Discrete names and types usually obey the order of the assigned integers. One hot encoding uses one feature entry for one discrete value. It implies nothing more than that the discrete values are different. The encoding methods are specified in Table 1. We choose to map the microarchitecture code names, memory types, and instruction set extensions into consecutive integers, with their natural order. For the SKU types, it is easier to do one hot encoding. We also distinguish the workloads using one hot encoding. Before training and testing, numerical and integer data need to be standardized, but one hot encoding data do not.

**Dynamic Workload Features:** Every entry in the SPEC and Geekbench datasets contains a configuration (SKU, current frequency, memory size), the name of the workload, and performance (relative runtime, which is the value to predict). We then get the CPU specifications from the Intel processor dataset by the SKU processor number. We do not use the scores provided in SPEC and Geekbench as performance metrics. For SPEC, we use the runtime in seconds. For Geekbench, we assume that the total amount of work is constant for a given workload and we use the inverse of throughput to estimate runtime. Details on the data preprocessing are in Section 3.2.

## 3.2 Data Preprocessing

The SPEC2006 repository provides runtime in seconds. Geekbench 3 provides throughput, e.g., as GB/s. We first average the performance of the workloads with the same (CPU SKU, Frequency, Memory Size) configuration. Variations of system software such as OS or compiler are not considered in this work. Geekbench 3 runtime is taken to be the inverse of throughput, assuming that one workload has a constant total amount of work. The average runtime is then converted into relative runtime for the chosen reference machine based on Equation (1). The specific choice of the reference machine does not affect the prediction. The relative performance is normalized to have mean 0 and standard deviation 1. The preprocessing steps are typical of machine learning experiments [22].

## 3.3 Model Selection

In this section, we explain our choice of DNNs and LR for this work. We then illustrate our approach for tuning DNN topology and the hyperparameters of DNN and LR.

**DNNs:** DNNs have proven successful in many domains, from regression and classification to game playing. Compared with traditional methods, DNN excels when data size is large, scales well with more data, and it does not require expertise in feature engineering. The advantages of DNN make it a very good fit for the performance prediction scenario. Thus, in this article, we use DNN as our predictive model.

The topology and hyperparameters are tuned through model selection. To do model selection, we randomly split the dataset into a training set, a validation set, and a testing set. The testing set is the last holdout set, to avoid using all data to do model selection. We start by sweeping the hyperparameters in all experiments, including the number of layers, the nodes per layer, the learning rate, the number of training epochs, the activation function, and the optimizer. The models are trained with the training set. The model with the lowest loss on the validation set is selected.

We find that the best set of hyperparameters is always the same. Therefore, in all experiments we use a DNN with three hidden layers and 100 nodes per layer. The learning rate is 0.001 over 300 training epochs. The activation function is tanh and the optimizer is RMSprop. Thus, for example, the number of weights for predicting SPEC is 50 million. Fortunately, training is offline and has a one-time cost.

**LR:** LR has been shown to work well in the CPU performance prediction literature [20], while DNN is known to be able to explore non-linear interactions between features, and it has been shown to be more accurate than LR at performance prediction [16, 17]. Unlike DNNs, LR has interpretable weight parameters. In realistic scenarios, people can choose LR or DNN based on the parameters, accuracy, and the value of interpretable weights. In order to make it a fair comparison, we reimplement LR with mean absolute error (MAE) as the loss function. That is, the model is optimized to yield the lowest MAE. Minimizing the MAE entails assuming the noise distribution to be Laplacian [4]. In our experiments, both DNN and LR have L1 regularization with a weight of 0.01. They take exactly the same training sets, testing sets, and features.

**Other Models**: Popular predictive models other than DNN include LR, support vector machines (SVMs), *k* nearest neighbors (kNNs), PCA, and genetic algorithms (GAs). In Section 7, we list studies using some of those models. However, SVM is not suitable for large-scale training because of its computational complexity [35]. kNN, PCA, and GA are better when there are fewer data points (<100) and simpler feature relations. Besides, traditional methods need carefully selected features, and accuracy does not benefit from more training data. Therefore, we choose to use DNN and compare it with one traditional method, LR.

## 3.4 Metrics

**MAE:** We use MAE as the measure of accuracy, rather than mean squared error (MSE). The reasons are as follows.

(1) There may be outliers and errors in the submitted data. With MSE, the optimizer penalizes the outliers much more than normal data. However, we do not want to overfit the model to the outliers [22].
(2) If a prediction is $x$ and the MAE of the model is 0.01, the real value of $x$ is very likely to be $x \pm 0.01$. MSE is not as easy to interpret.

Because the MAE is in a standardized space, we use two baselines to help understand it. We assume two independent and identically distributed random variables, $a$ and $b$, from Gaussian distribution N(0, 1). The expected distance between $a$ and $b$ is $\frac{2}{\sqrt{\pi}}$, which is approximately 1.13 [32]. Therefore, a prediction with MAE of 0.25 is a very good prediction. Another way to interpret the MAE is to compare it with the standard deviation of the testing set. A naive way to do prediction is to always predict the mean of the dataset. This way, by definition the MAE is the standard deviation

$$\text{MAE} = \frac{\sum_{i=0}^{n} |x_i - m|}{n} = \text{Standard Deviation}, \tag{2}$$

where $n$ is the number of data points, $m$ is the mean of the testing set, and $x_i$ is the real value of predicted performance, after normalization. Note that while the whole dataset is normalized to have mean 0 and standard deviation 1, the testing sets may be in different scales. In the captions of Figures 6 and 7, we compare the MAEs with the standard deviations of the testing sets, and show that the MAEs are much smaller. However, what we just discussed are naive baselines. The next metric can show that our prediction is good enough in realistic scenarios.

**Top-*k* Accuracy:** To bring the MAE metric into realistic context, we use the predictive model to help select SKUs. We predict performance and use that prediction to rank the SKUs.

We choose top-*k* accuracy, which measures whether the known best SKU is among the top *k* predicted SKUs. It is a popular metric for image classification tasks [13, 19]. Other metrics for comparing rankings include Pearson correlation, Spearman footrule distance, and Kendall rank correlation. They measure the difference between the tested ranking and the expected ranking, by considering every item in each ranking. Top-$k$ accuracy is a better metric for this work, because users care more about whether the best SKU is in the top $k$ choices, than about how similar one SKU ranking is to another. In this context, top-$k$ accuracy is more interpretable than the ranking correlations.

We construct three baselines to compare with our predictive model. Without the predictive model, it is customary to compare microarchitecture code names (newer is better), frequencies (higher is better), and cache sizes (larger is better). Thus, we construct the baselines taking those factors into consideration. Though the baselines are simple, this is how most customers make purchasing decisions.

A **Frequency:** Rank the SKUs based on frequencies. If frequencies are the same, rank them with cache sizes.
B **Cache:** Rank the SKUs based on cache sizes. If cache sizes are the same, rank them with frequencies.
C **Frequency + Cache:** Rank the SKUs based on a summation of frequency and cache size with weights. The weights are discussed in Section 5.1. If the values are the same, rank them with microarchitecture code names.

To measure the top-$k$ accuracy, we shuffle our training or testing set, find the best SKU in the testing set using real performance from the datasets, and measure how often the best SKU is one of the top $k$ SKUs as ranked by our predictive model and the baselines.

## 4 CASE STUDIES OVERVIEW

In this section, we sketch the case studies, including prediction for new SKUs and for new workloads, and cross-prediction between suites. We implement our models with Keras (https://keras.io).

### 4.1 Case 1: Prediction for New SKUs

In this case, we show that we can predict the performance of SPEC and Geekbench workloads on new SKUs. Using our open-source model, trained with SPEC and Geekbench data for existing SKUs from online repositories, users can predict Geekbench and SPEC performance on new SKUs of interest and choose the ones likely to perform the best.

In order to test the model on new SKUs, we adopt repeated random sub-sampling validation, a widely used cross-validation method [8]. We hold out several SKUs from the dataset, train the model with the remaining data, and test the model with the holdout set.

We randomly choose 10% of the SKUs from one microarchitecture at a time as our testing set. We use the rest of the data as our training set. We repeat this process 20 times for every microarchitecture. We choose 20 repetitions after trying 50 and finding that the results were statistically consistent. Comparing the MAEs of 20 DNN repetitions versus 50, the maximum absolute difference is 0.005 and the standard deviation difference is 0.009. Both are minimal.

The premise of this case study is that the predictive models are able to learn the performance function of a workload using the CPU specifications in Table 1. Specifically, the training set may contain SKUs with the same microarchitecture as the new SKU, and SKUs with similar features such as L3 cache, frequency, and memory size, but different microarchitecture. The predictive model then predicts the new SKU's performance as though interpolating the training set.

Note that the prediction is nontrivial, even if the same microarchitecture or the full feature set of a different microarchitecture is in the training set. As shown in Figure 2, both microarchitecture and other SKU features (such as frequency) introduce large variations in performance. Knowing one of them does not make the prediction easy.

### 4.2 Case 2: Prediction for New Workloads

In this study, we show that after measuring performance of a new workload on just a handful of SKUs, we can train a model to predict the workload's performance on other SKUs. Armed with such predictions, consumers can choose the best processor for the workload based on both performance and price.

To demonstrate the approach, we split our dataset, taking out one workload at a time and treating it as the new workload. Then we train the model with all the other workloads' data plus a certain amount of data from the chosen workload. We perform the experiment separately for SPEC and Geekbench. We refer to this case as self-prediction of benchmark suites.

The premise of this and the next case study is that, given the data of a new workload on several SKUs, the model can find workloads with similar performance on those SKUs, and use the performance of those workloads to predict the new workload's performance on other SKUs. Therefore, the prediction accuracy should be related to the similarity of the new workload to the workloads in the training set.

For every workload A, we remove the data for workload A from the overall dataset. We call the remaining data *data_rest*. We then sample a fixed set of SKUs for workload A as a testing set, and we refer to the rest of the data for workload A as *train_pool*. We randomly pick $n$ SKUs from *train_pool* (where $n$ is 1, 5, 10, or 50) and combine their data with *data_rest* to form a training set. We repeat the process for 20 iterations.

### 4.3 Case 3: Cross-Prediction Between Suites

The experiments in case 2 are conducted within each benchmark suite. Case 3 is an extension of case 2 that we call cross-prediction. We predict one workload in Geekbench using all data for SPEC plus that collected using several SKUs for the chosen workload. Then we repeat with Geekbench and SPEC switched.

For every workload A in Geekbench, we split the whole dataset into the data of workload A and the rest. We then sample a fixed set of SKUs from workload A as a testing set, and we refer to the remaining data of workload A as *train_pool*. We randomly pick $n$ SKUs from *train_pool* and combine their data with the SPEC data as a training set. We sweep $n$ over 1, 5, 10, and 50, and we repeat the process 20 times.

## 5 PREDICTION ACCURACY

In this section, we show prediction accuracy measured byMAE in the case studies. We show that DNN is more accurate than LR, and case 2 (self-prediction) is more accurate than case 3 (cross-prediction).

### 5.1 Case 1: Prediction for New SKUs

Figure 6 compares the MAEs of LR and DNN for SPEC and Geekbench, respectively. The heights of the bars represent the average MAEs of the 20 random test sets. The error bars show one standard deviation of the MAEs. A smaller error bar indicates that different selections of testing set give large differences in MAEs, therefore the model is more robust to predict different SKUs.

For both SPEC and Geekbench, DNN always has lower MAEs and is less sensitive to the selection of the test set than LR. That means the relationships between features and performance are not simply linear. In Figure 6 (bottom), LR has a very large MAE and standard deviation when predicting Geekbench on Broadwell SKUs. That is because in Figure 3 (bottom), there are only five Broadwell SKUs, and one is desktop while the others are servers. Using servers to predict desktop performance leads to large MAEs for LR. Apparently, DNN addresses this better. To compare the

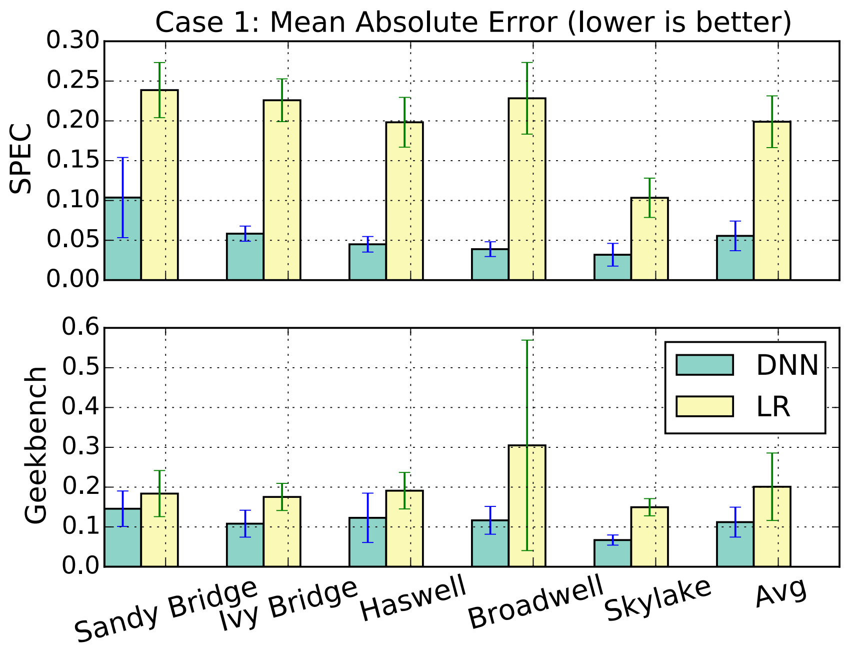

Fig. 6. Results of predicting the performance of SPEC (top) and Geekbench (bottom) on new SKUs. We compare linear regression (LR) and deep neural network (DNN) models. DNN always has lower mean absolute errors (MAEs) than LR. With DNN, the average MAE for SPEC is 5.5% and for Geekbench it is 11.2%. The standard deviations of the test sets are all over 20%.

average MAEs (the last set of bars in Figure 6), LR is 19.9% and DNN is 5.5% for SPEC, and LR is 20% and DNN is 11.2% for Geekbench. DNN is a better choice when the dataset is large and interactions between features are complicated.

The average MAE is 5.5% for SPEC and 11.2% for Geekbench using DNN. The MAE for Geekbench is higher than SPEC because Geekbench users are not like SPEC users, who are mostly computer architects and system engineers, benchmarking on dedicated machines. One can easily install Geekbench and produce scores on personal devices. Results may be noisy if other applications are running at the same time, causing contention in computing and memory resources. We observe larger variance in Geekbench data. We compute the average standard deviation of the performance for the same workload with the same (SKU, frequency, memory) configuration. It is 4.4% for Geekbench, and 1.1% for SPEC. The larger noise from the data makes it harder to predict Geekbench.

The accuracy of simulation-based prediction is about 4% [20], which is slightly lower than our 5.5% MAE of SPEC. Compared to the simulation-based approach, the extra noise in our approach comes from the performance variation introduced by compilers and operating systems in the datasets, and no architectural features. We use no architectural features which makes the prediction harder, but it also makes the method easier to apply to any other public datasets, without the need for tedious simulation.

### 5.2 Case 2: Prediction for New Workloads

In this section, we show the prediction results of case 2. We show that the DNN prediction accuracy is largely determined by the similarity of the predicted workload to the rest of the benchmark

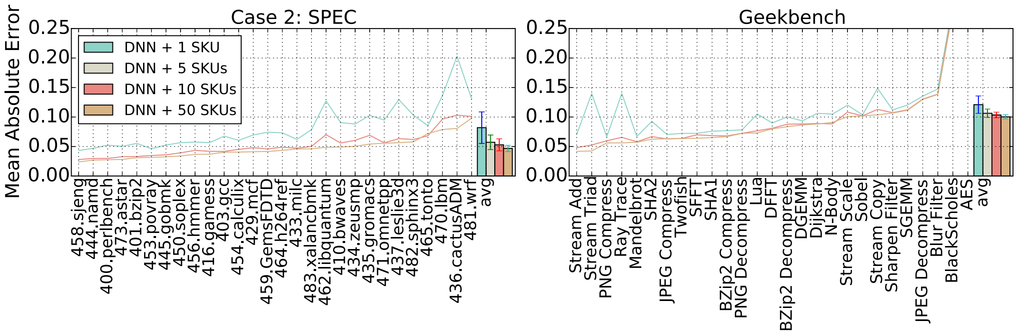

Fig. 7. The results of case 2. The workloads are sorted by the MAE after adding 50 SKUs. The workloads on the right are outliers. The average MAEs are shown in the bars on the right. The line for adding 5 SKUs is omitted, but the average bar for that case is shown. The standard deviations of the testing sets are all over 11%.

suite. We quantify the outlierness of a workload and show that it is positively correlated with the prediction MAE, with a Pearson correlation of 0.69.

**Prediction Results:** The results of SPEC are shown in Figure 7 (left). The workloads are sorted based on the MAEs after adding 50 SKUs (brown line). (The line for 5 added SKUs is omitted but the average bar is shown.) The standard deviations of all the testing sets are larger than 20%, and our MAEs in all cases are lower than the standard deviation. For 5, 10, and 50 added SKUs, the MAEs are 5.7%, 5.3%, and 4.7%, respectively. MAE improvements when moving from 10 SKUs to 50 are similar to those when moving from 5 SKUs to 10. Since adding 40 SKUs is more difficult than adding 5, we conclude that using 10 SKUs for the new workload is a reasonable choice.

Similarly, the Geekbench prediction results are shown in Figure 7 (right). The workloads are sorted and the rightmost workloads are outliers with very high MAEs. The standard deviations of Geekbench testing sets are all above 11%. The outliers' standard deviations are at least twice their MAEs. That means our model shrinks the confidence interval by at least 50%.

Case 2 is harder than case 1. In case 1, the average MAE is 11%. In case 2, the average MAE is 14.2%, with two obvious outlier workloads (Blackscholes and AES). We conclude that case 2 works reasonably well, apart from the two outliers. Also, for the same reason as with SPEC, we choose 10 as the proper number of SKUs to use.

**Insights on Benchmark Similarity:** By studying MAEs across SPEC workloads, we find that the prediction MAE is correlated with the similarity of the predicted workload to the rest of the benchmark suite. This observation supports the statement in Section 4.2 that the model finds similar workloads based on the data points of the new workload, and it uses the similar workloads to predict the performance of the new workload on other SKUs.

In Figure 7 (left), after adding 50 SKUs it is still relatively hard to predict the workloads on the right. We call these workloads outliers. The workload with highest MAE is 481.wrf and it is also the outlier lying above all other workloads in Figure 5, which shows the workloads' relative performance scaling in the varied SKU configurations.

Some outliers are well-studied in prior works. Phansalkar et al. show that 436.cactusADM is an outlier in terms of memory access characteristics in Figure 9 of [24]. However, workload features collected in prior works are unable to explain why 481.wrf is such an outlier in SKU scaling space, as shown in Figures 7 and 5. Future work will try to answer that question.

To study the outlierness of the workloads in Figure 7 (left), we ran a $k$-means algorithm to split the workloads into two clusters, *cluster1* and *cluster2*. Their centroids $C_1$ and $C_2$ are plotted as stars

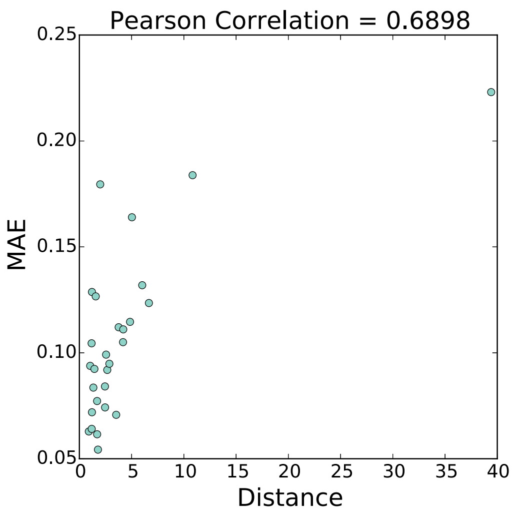

Fig. 8. Correlation of SPEC workloads' outlierness (distance) with MAE after adding 50 SKUs. The distance of a workload is its distance from the centroid in its cluster. Centroids are in Figure 5.

in Figure 5. We compute the distance of a workload to its cluster's centroid as

$$\begin{aligned} distance &= I(x \in cluster1)||x - C_1||_2 \\ &\quad + I(x \in cluster2)||x - C_2||_2, \end{aligned} \tag{3}$$

where $x$ stands for the location of the workload in Figure 5 and $I$ yields 1 if its argument is true and 0 otherwise. The distance quantifies the outlierness of a workload.

Figure 8 plots a workload's MAE after 50 added SKUs ($y$-axis) against its distance ($x$-axis). Distance and MAE have a Pearson correlation of 0.6898. We can use distance to identify workloads with high errors. Workloads with distance higher than 5 have MAEs higher than 10%. This correlation explains why the outliers have higher MAEs than others even after adding 50 SKUs.

**Conclusion:** This case shows that we are able to predict a new workload's performance by running it on 10 SKUs. The accuracy of the prediction depends on the similarity of the workload to those in the training set. Even if it is very different, our model is still able to shrink the confidence interval.

## 5.3 Case 3: Cross-Prediction Between Suites

Figure 9 shows the results of cross-prediction: using Geekbench to predict SPEC (top) and *vice versa* (bottom). Because the MAEs of the workloads and the outliers are similar to case 2 (self-prediction), we only show the average MAE bars in this case.

To compare self-prediction and cross-prediction, we first compare Figure 9 (top) with the bars in Figure 7 (left), both of which predict SPEC workloads. Cross-prediction gives higher MAEs, for any number of added SKUs. After adding 50 SKUs, the average MAE of cross-prediction is 16.9%; the average MAE of self-prediction is 4.6%. Comparing the predictions of Geekbench in Figure 9 (bottom) and the bars in Figure 7 (right), after adding 50 SKUs the average MAE for cross-prediction is 12.6% and that for self-prediction is 10%.

Cross-prediction gives consistently higher errors because the workloads in SPEC and Geekbench are very different, more different than the workloads in either benchmark suite itself. In other words, the workloads in a benchmark suite are more similar to each other than to workloads outside of the benchmark suite.

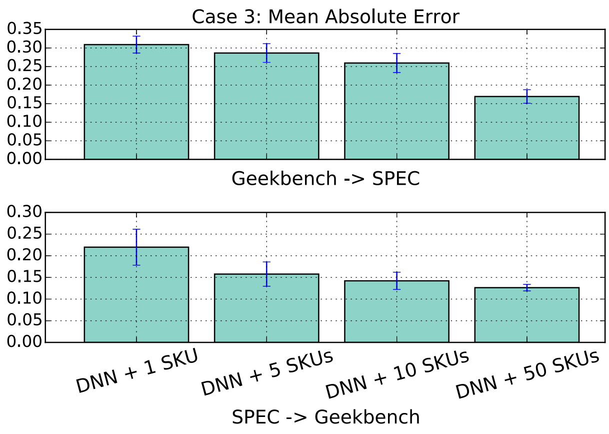

Fig. 9. Case 3 is about cross-prediction. SPEC workloads are predicted with the Geekbench dataset (top). Geekbench workloads are predicted with the SPEC dataset (bottom). Cross-prediction has higher errors than self-prediction (case 2).

Benchmark suites like SPEC CPU2006 are expected to be diverse and representative of real-world workloads. However, a benchmark suite is developed for a purpose, and different suites usually have different purposes. The purpose itself is the reason why the workloads in the suite are similar. The SPEC CPU2006 benchmark suite is intended for computer designers and architects to use for pre-silicon design analysis. SPEC workloads are drawn from and representative of real workloads.

In contrast, Geekbench is intended to stress a certain part of the hardware. For example, memory microbenchmarks do a lot of memory streaming operations to study the bandwidth limit of the hardware system. From this point of view, it is reasonable that SPEC and Geekbench are very different and the workloads in each suite are more similar.

**Conclusion:** Cross-prediction is harder than self-prediction. To predict performance for new workloads, one needs a training set with a very diverse collection of workloads. Benchmark suites tend to be self-similar, because each serves a specific purpose. Even so, cross-prediction is useful, as shown in the next section.

## 6 SKU RANKING COMPARISON

In this section, we use the performance prediction results of the DNN model to rank SKUs. We then compare the SKU rankings with the baseline metrics described in Section 3.4. The baselines represent the customary ways of making purchasing decisions. Our methodology makes SKU selection more accurate without requiring expertise on workload characterization or computer architecture specifications.

### 6.1 Case 1: Prediction for New SKUs

In this section, we show that our DNN model outperforms the three baselines in Section 3.4 when using SPEC and Geekbench to select new SKUs.

The results of SPEC and Geekbench are shown in Figure 10. For baseline C, a combination of frequency and cache, we swept 100 combinations of weights. We combined $frequency(GHz) \times w$ with $cache(MB) \times (1 - w)$, and swept $w$ from 0.01 to 1, with a step size of 0.01. In Figure 10, we use $0.9 \times frequency + 0.1 \times cache$ so that the frequency and cache values are in about the same range. The top-3 accuracy of baseline C combining frequency and cache is no better than A (frequency) or B (cache).

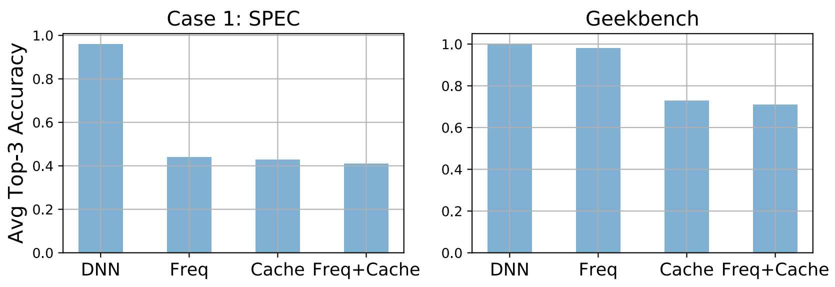

Fig. 10. SKU ranking results of case 1. Our DNN model has the highest top-3 accuracy.

Frequency is the best metric among the three baselines. The following subsections show similar results, therefore we show baseline A and drop the others in the rest of the article.

Our predictive model outperforms all of the workload-indifferent baselines. For Geekbench, the frequency metric achieves accuracy comparable to that of our DNN model. The Geekbench suite as a whole is quite frequency-sensitive. The SPEC suite, on the other hand, is sensitive to neither frequency nor cache, which suggests that choosing SKUs for complicated workloads requires more than simply comparing frequency or cache.

**Conclusion:** When selecting a SKU that optimizes SPEC or Geekbench performance, our predictive model works better than simply comparing frequencies, cache sizes, or a combination of the two. Selecting a SKU by frequency is better than by cache size, and is more successful for microbenchmarks like Geekbench than for complicated workloads like SPEC.

## 6.2 Cases 2 and 3: Self- and Cross-Prediction

In this section, we compare the top-$k$ accuracies of cases 2 and 3 and baseline A (frequency). Results using baselines B and C are consistent with those presented, so we omit them. We show case 2 and case 3 in the same plots to compare their results. Both cases use 10 additional SKUs for the new workload. The results appear in Figure 11.

It is very difficult to find the best SKU for SPEC workloads for cases other than SPEC self-prediction (case 2). The top-3 accuracy of case 3 is very small, and that of frequency is 0. We start to observe more accuracy from them when we relax the choices to top-7 SKUs. It indicates building the predictive model is the only reasonable way to find good SKUs for SPEC. The situation can get worse for the workloads in people's daily lives such as web browser, Microsoft Office suite, Adobe suite, and other customized workloads from individual companies.

Self-prediction has higher accuracies than cross-prediction for most of the workloads, suggesting self-prediction is easier. The ranking accuracies further support what has been observed from the prediction accuracy studies in Figures 7 and 9, and the self-similarity statement in Section 5.3. Exceptions are AES, Dijkstra, and Stream Scale from Geekbench, which have higher accuracies when trained with SPEC. Among them, AES is the outlier with the highest MAE from the self-prediction accuracy results in Figure 7 (right). The results indicate that those three workloads are more similar to SPEC in terms of selecting the best SKUs.

Self- and cross-prediction are more accurate than frequency. Note that in cases 2 and 3, the test set is fixed, and the only variation in the 50 iterations is the random additional 10 SKUs from the new workload, as described in Section 4. Therefore, for frequency bars in Figure 11, there is only one ranking of SKUs to test, and the top-$k$ accuracy is either 0 or 1, depending on whether the best SKU is ranked top-$k$ or not. By averaging the accuracies for the whole suite, we find that frequency performs the worst. Though the previous subsection shows the accuracy of frequency

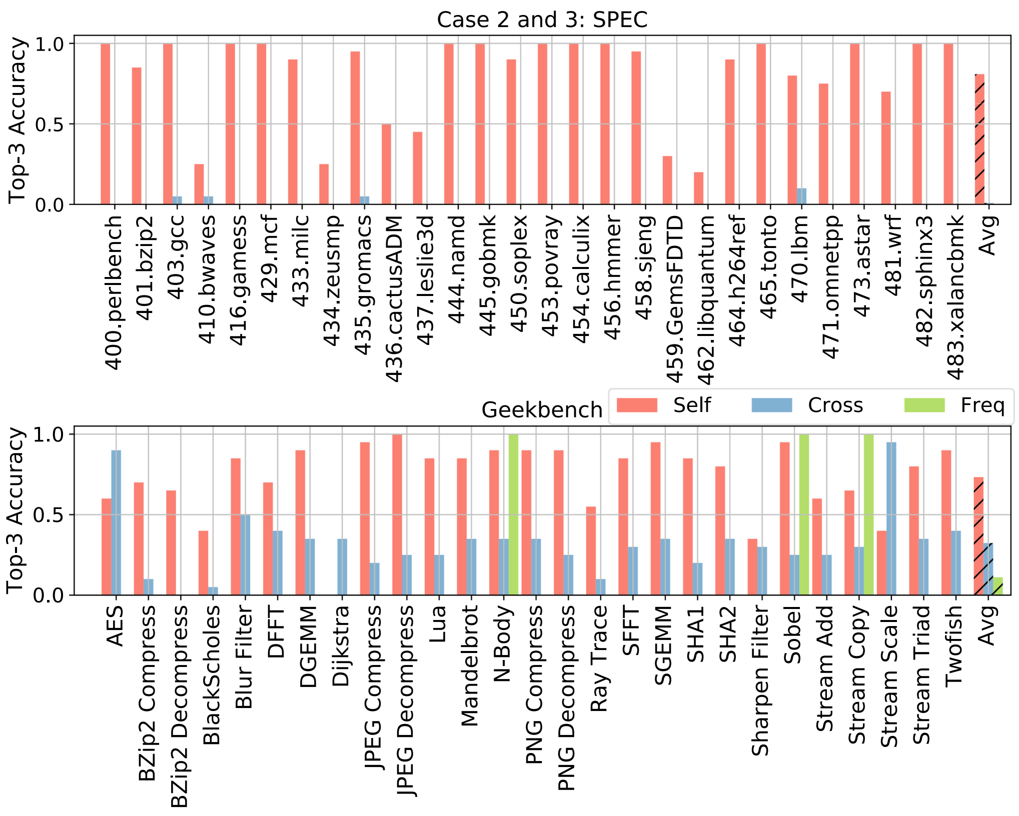

Fig. 11. Based on SPEC (top) and Geekbench (bottom), compare the SKU rankings by self-prediction (case 2), cross-prediction (case 3), and the best baseline (Frequency). Self-prediction performs the best. Frequency is not a reasonable approach to select SKUs for new workloads.

to be comparable to DNN prediction for Geekbench (Figure 10), individual workloads may or may not be sensitive to frequency. Frequency is not a good way to select SKUs.

**Conclusion:** SPEC self-prediction is the only reasonable way to find the best SKU for SPEC workloads. Self-prediction is easier than cross-prediction in most cases. Frequency has the lowest accuracy, so frequency alone should not be the deciding factor when selecting SKUs for new workloads.

## 6.3 Benchmark Suite Comparison

In this section, we compare the benchmark suites in terms of SKU ranking. While there is no prediction involved, it is natural to compare the two benchmark suites after comparing the individual workloads in them.

Geekbench is very popular on computer enthusiast websites, with many articles claiming that "Geekbench suggests A outperforms B." Such articles affect many consumers' choices when purchasing computing products. Most people use laptops to run workloads such as web browsers, Microsoft's Office suite, Adobe software, and so on. We describe these as realistic workloads. Consumers choose CPUs based on Geekbench scores because they assume that Geekbench predicts the performance of realistic workloads. Here we view SPEC workloads as falling between realistic workloads and Geekbench in terms of complexity and real-world relevance.

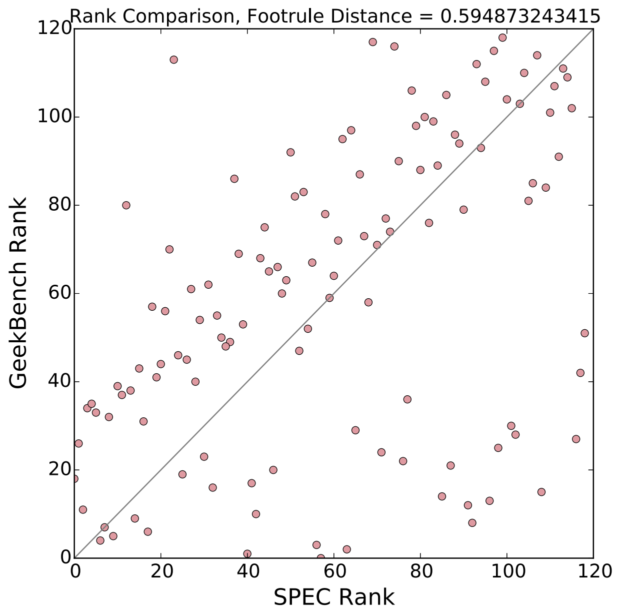

Fig. 12. Comparison of SKU rankings based on SPEC and Geekbench averages. Points are clustered above the diagonal because Geekbench ranks SKUs lower than SPEC (giving them higher ranking indexes). The horizontal cluster below shows that Geekbench poorly distinguishes SKUs with quite different SPEC behavior.

To show how the two suites produce different rankings, we rank the SKUs that SPEC and Geekbench datasets have in common by averaging each benchmark suite's performance on each such SKU. Figure 12 shows the ranking comparison. The $x$-axis is the rank based on SPEC and the $y$-axis is the same SKU's rank based on Geekbench. The footrule distance between the two rankings is 0.59. (The range is from 0 to 1.) If the ranks were consistent on both suites, the points would lie along the gray line ($x = y$). We observe two patterns in Figure 12. One is the cluster of points above the gray line; Geekbench ranks those SKUs lower than SPEC (gives them higher ranking indexes). Another pattern is the cluster of points below the gray line and close to the $x$-axis. SPEC ranks those SKUs from 0 to 120, while Geekbench ranks them from 0 to around 50. In other words, the SKUs have very different SPEC performance, but are poorly differentiated by Geekbench.

The Geekbench SKU ranking is not consistent with SPEC rankings. That means that if an enthusiast article says "Geekbench shows that A outperforms B," SPEC may suggest that B outperforms A. It is very unlikely that a Geekbench score can give accurate indications for realistic workloads either. If one buys a CPU configuration based on its highest Geekbench score, the hardware may not be the best for running the workloads that he or she runs daily.

This suggests that people use Geekbench incorrectly. The aggregate score of a benchmark suite is misleading. Geekbench is designed to stress floating point, integer, and memory operations. It makes more sense to compare the performance of individual microbenchmarks that are relevant to realistic workloads. If one runs a lot of matrix operations and image processing workloads, such as MATLAB and Adobe Photoshop, the floating point benchmarks in Geekbench are very helpful.

**Conclusion:** Aggregate Geekbench scores are misleading. Benchmarking relevant individual Geekbench workloads makes more sense. SKU rankings by Geekbench are inconsistent with rankings by SPEC, and probably also with rankings for realistic workloads. For a non-expert consumer, the best way is to take all datasets available, plus the new workload's performance on 10 SKUs, and train a predictive model as in cases 2 and 3.

## 7 RELATED WORK

### 7.1 Performance Prediction

For performance prediction, our work uses DNNs, regarded as the state-of-the-art machine learning algorithm. It has been proven to be able to simulate any function [5].

Unlike simulation-based performance prediction [16, 17, 20, 28], our approach does not use microarchitecture features such as instruction issue width, read-write buffer size, or number of pipeline stages. We use only the CPU specifications that manufacturers provide. Microarchitecture features would make performance prediction easier and more accurate, but our approach is more helpful for consumers choosing CPUs. Even when manufacturers describe microarchitecture features, consumers need expert knowledge to make use of such information.

Performance prediction is a well-studied field. There are mechanistic models [3, 10–12, 18, 33, 34], empirical models [9, 21, 31], and machine learning models [1, 36]. Zheng et al. used performance counters and a "stochastic dynamic coupling" heuristic for aligning host and target execution samples to estimate performance using only binary code [37]. In contrast, we focus on CPU SKU selection for consumers, rather than performance prediction for hardware/architecture designers. Our method does not use performance counters or architecture features other than those accessible to all consumers.

Piccart et al. use data transposition to rank commercial machines [27]. Their work exploits machine similarity rather than workload similarity. Our work has different objectives and uses different metrics. It exploits workload similarity and benchmark suite self-similarity to help users pick the best CPU SKU for their needs.

### 7.2 Workload Characterization

Phansalkar et al. develop a series of SPEC workload characterizations [23–26]. However, they do not focus on workload performance scaling as we do, so their classification results are not consistent with ours. Our approach provides an orthogonal way to characterize workloads.

Some researchers use static workload features to predict performance. Shelepov et al. use spatial locality, cache size, and frequency to predict speedup on a variety of processors [29, 30]. Hoste et al. collect microarchitecture-independent features to classify workload performance [14, 15]. Delimitrou et al. study workload characteristics for cloud scheduling [6, 7]. In our work, we predict a new workload's performance on hundreds of CPU SKUs. The number of workloads is much smaller than the number of hardware platforms. Because the workload features are static, we argue that it is hard to use static features in this work.

## 8 CONCLUSION

To help customers select proper CPU SKUs and overcome the limitations of public benchmark datasets, we have presented statistical and predictive analysis of workload performance, with data collected from the SPEC CPU and Geekbench 3 suites and Intel processor specifications.

We demonstrated performance prediction for *new SKUs* and *new workloads* with DNNs. For new SKUs, the accuracy is 5% (SPEC) and 11% (Geekbench) MAE. With the same accuracy, we find that we can predict a new workload's performance by running it on 10 SKUS. We compared our predictive methods against workload-indifferent metrics for selecting processors. Our predictive model is the only approach that achieves reasonable accuracy in all three case studies. Notably, workload-indifferent methods of SKU selection do not work for new workloads.

For the first time in the literature, we showed benchmark suite self-similarity quantitatively by cross-prediction and comparison of SPEC and Geekbench SKU rankings. The accuracy of cross-prediction is lower than that for prediction within the suites because they are so different: 25.9% mean error to predict SPEC and 14.2% to predict Geekbench.

Rankings based on average Geekbench are inconsistent with those based on average SPEC. Geekbench rankings do not imply SPEC rankings. It is also hard to draw any conclusions about realistic workloads from a benchmark suite. We suggest studying the Geekbench or SPEC workloads of interest, rather than relying on aggregate scores.

## ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their thoughtful comments and constructive suggestions which significantly improved the quality of this article. The authors would like to thank Bob Adolf, Udit Gupta, Glenn Holloway, Svilen Kanev, Lillian Pentecost, Saketh Rama, Brandon Reagen, Margo Seltzer, Sophia Shao, and Sam Xi for their feedback.

## REFERENCES

[1] Newsha Ardalani, Clint Lestourgeon, Karthikeyan Sankaralingam, and Xiaojin Zhu. 2015. Cross-architecture performance prediction (XAPP) using CPU code to predict GPU performance. In *Proceedings of the 48th International Symposium on Microarchitecture.* ACM, 725–737.

[2] Simone Campanoni, Kevin Brownell, Svilen Kanev, Timothy M. Jones, Gu-Yeon Wei, and David Brooks. 2014. HELIX-RC: An architecture-compiler co-design for automatic parallelization of irregular programs. In *Proceedings of the 41st Annual International Symposium on Computer Architecuture (ISCA'14).* 217–228.

[3] Xi E. Chen and Tor M. Aamodt. 2011. Hybrid analytical modeling of pending cache hits, data prefetching, and MSHRs. *ACM Transactions on Architecture and Code Optimization (TACO)* 8, 3 (2011), 10.

[4] Vladimir Cherkassky and Yunqian Ma. 2004. Comparison of loss functions for linear regression. In *Proceedings of the 2004 IEEE International Joint Conference on Neural Networks.* Vol. 1. IEEE, 395–400.

[5] George Cybenko. 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems* 2, 4 (1989), 303–314.

[6] Christina Delimitrou and Christos Kozyrakis. 2013. Paragon: QoS-aware scheduling for heterogeneous datacenters. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'13)*, Vol. 48. ACM, 77–88.

[7] Christina Delimitrou and Christos Kozyrakis. 2014. Quasar: Resource-efficient and QoS-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'14).* ACM, 127–144.

[8] Werner Dubitzky, Martin Granzow, and Daniel P. Berrar. 2007. *Fundamentals of Data Mining in Genomics and Proteomics.* Springer Science & Business Media.

[9] Phillip Ein-Dor and Jacob Feldmesser. 1987. Attributes of the performance of central processing units: A relative performance prediction model. *Communications of the ACM* 30, 4 (1987), 308–317.

[10] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E. Smith. 2009. A mechanistic performance model for superscalar out-of-order processors. *ACM Transactions on Computer Systems (TOCS)* 27, 2 (2009), 3.

[11] Stijn Eyerman, Kenneth Hoste, and Lieven Eeckhout. 2011. Mechanistic-empirical processor performance modeling for constructing CPI stacks on real hardware. In *Proceedings of the 2011 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'11).* IEEE, 216–226.

[12] Allan Hartstein and Thomas R. Puzak. 2002. The optimum pipeline depth for a microprocessor. In *Proceedings of the 29th Annual International Symposium on Computer Architecture (ISCA'02).* IEEE Computer Society, 7–13.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 770–778.

[14] Kenneth Hoste, Lieven Eeckhout, and Hendrik Blockeel. 2007. Analyzing commercial processor performance numbers for predicting performance of applications of interest. In *Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems.* ACM, 375–376.

[15] Kenneth Hoste, Aashish Phansalkar, Lieven Eeckhout, Andy Georges, Lizy K John, and Koen De Bosschere. 2006. Performance prediction based on inherent program similarity. In *Proceedings of the 2006 International Conference on Parallel Architectures and Compilation Techniques (PACT'06).* IEEE, 114–122.

[16] Engin Ipek, Bronis R. De Supinski, Martin Schulz, and Sally A. McKee. 2005. An approach to performance prediction for parallel applications. In *European Conference on Parallel Processing*. Springer, 196–205.

[17] Engin İpek, Sally A. McKee, Rich Caruana, Bronis R. de Supinski, and Martin Schulz. 2006. Efficiently exploring architectural design spaces via predictive modeling. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'06).* ACM, 195–206.

[18] Tejas S. Karkhanis and James E. Smith. 2004. A first-order superscalar processor model. In *Proceedings of the 31st Annual International Symposium on Computer Architecture, 2004*. IEEE, 338–349.

[19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Proceedings of Advances in Neural Information Processing Systems (NIPS'12)*. 1097–1105.

[20] Benjamin C. Lee et al. 2006. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 185–194.

[21] Ang Li, Xuanran Zong, Srikanth Kandula, Xiaowei Yang, and Ming Zhang. 2011. CloudProphet: Towards application performance prediction in cloud. In *Proceedings of the ACM SIGCOMM 2011 conference (SIGCOMM'11)*, 426–427.

[22] Kevin P. Murphy. 2012. *Machine Learning: A Probabilistic Perspective*. MIT Press.

[23] Aashish Phansalkar, Ajay Joshi, Lieven Eeckhout, and Lizy Kurian John. 2005. Measuring program similarity: Experiments with SPEC CPU benchmark suites. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'05)*. IEEE, 10–20.

[24] Aashish Phansalkar, Ajay Joshi, and Lizy K. John. 2007. Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*. ACM, 412–423.

[25] Aashish Phansalkar, Ajay Joshi, and Lizy K. John. 2007. Subsetting the SPEC CPU2006 benchmark suite. *ACM SIGARCH Computer Architecture News* 35, 1 (2007), 69–76.

[26] Aashish Shreedhar Phansalkar. 2007. *Measuring Program Similarity for Efficient Benchmarking and Performance Analysis of Computer Systems*. The University of Texas at Austin.

[27] Beau Piccart, Andy Georges, Hendrik Blockeel, and Lieven Eeckhout. 2011. Ranking commercial machines through data transposition. In *Proceedings of the 2011 IEEE International Symposium on Workload Characterization (IISWC'11)*. IEEE, 3–14.

[28] Sameh Sharkawi, Don Desota, Raj Panda, Rajeev Indukuru, Stephen Stevens, Valerie Taylor, and Xingfu Wu. 2009. Performance projection of HPC applications using SPEC CFP2006 benchmarks. In *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS'09)*. IEEE, 1–12.

[29] Daniel Shelepov, Juan Carlos Saez Alcaide, Stacey Jeffery, Alexandra Fedorova, Nestor Perez, Zhi Feng Huang, Sergey Blagodurov, and Viren Kumar. 2009. HASS: A scheduler for heterogeneous multicore systems. *ACM SIGOPS Operating Systems Review* 43 (2009), 66–75.

[30] D. Shepelow and A. Fedorova. 2008. Scheduling on heterogeneous multicore processors using architectural signatures. In *Proceedings of the WIOSCA Workshop of the 35th Annual International Symposium on Computer Architecture (ISCA'08)*.

[31] Karan Singh, Engin İpek, Sally A. McKee, Bronis R. de Supinski, Martin Schulz, and Rich Caruana. 2007. Predicting parallel application performance via machine learning approaches. *Concurrency and Computation: Practice and Experience* 19, 17 (2007), 2219–2235.

[32] Benjamin Thirey and Randal Hickman. 2015. Distribution of Euclidean distances between randomly distributed Gaussian points in n-space. *arXiv:1508.02238*.

[33] Sam Van den Steen, Sander De Pestel, Moncef Mechri, Stijn Eyerman, Trevor Carlson, David Black-Schaffer, Erik Hagersten, and Lieven Eeckhout. 2015. Micro-architecture independent analytical processor performance and power modeling. In *Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS'15)*. IEEE, 32–41.

[34] Sam Van den Steen, Stijn Eyerman, Sander De Pestel, Moncef Mechri, Trevor E. Carlson, David Black-Schaffer, Erik Hagersten, and Lieven Eeckhout. 2016. Analytical processor performance and power modeling using micro-architecture independent characteristics. In *IEEE Transaction on Computers* 65, 12 (Dec. 2016), 3537–3551.

[35] Yuxuan Wang and DeLiang Wang. 2013. Towards scaling up classification-based speech separation. *IEEE Transactions on Audio, Speech, and Language Processing* 21, 7 (2013), 1381–1390.

[36] Xinnian Zheng, Lizy K. John, and Andreas Gerstlauer. 2016. Accurate phase-level cross-platform power and performance estimation. In *Proceedings of the 2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC'16)*. IEEE, 1–6.

[37] Xinnian Zheng, Haris Vikalo, Shuang Song, Lizy K. John, and Andreas Gerstlauer. 2017. Sampling-based binary-level cross-platform performance estimation. In *Proceedings of the Conference on Design, Automation & Test in Europe*. European Design and Automation Association, 1713–1718.

Received February 2018; revised August 2018; accepted September 2018