**MLTUNE: A Tool-chain for Automating the Workflow of Machine-Learning Based Performance Tuning**

[Extended Abstract]

Biplab Kumar Saha  
Computer Science Dept.  
Texas State University  
b_s183@txstate.edu

Saami Rahman  
Computer Science Dept.  
Texas State University  
saami.rahman@txstate.edu

Apan Qasem  
Computer Science Dept.  
Texas State University  
apan@txstate.edu

1. INTRODUCTION

The use of machine learning (ML) techniques has emerged as a promising strategy for performance tuning of HPC applications. Sophisticated predictive models have been developed for code optimization, workload characterization and program analysis. Current state-of-the-practice maintains that learning algorithms be trained on the developer platform and the pre-built models be embedded within a software tool (e.g., compiler [4] or autotuner [3]) before being shipped to the end-user. This practice imposes an inherent limitation on the predictive capabilities of the models. Since program behavior is intricately tied to characteristics of the target architecture, the accuracy of the models tend to be sensitive to variations in parameters of the underlying platform. Even a small change in the processor configuration can render some models ineffective. As such, it is imperative that the learning occur in the target environment.

Training a model on a new platform is a non-trivial task, however. The workflow is fairly extensive and requires knowledge of machine learning and statistical techniques. In addition to data collection and clean-up, feature extraction and selection must be done anew for each platform. For classifiers the training data must be labeled where labels may be a priori unknown. Finally, a suitable algorithm has to be selected based on the characteristics of the training data set. For application developers and performance engineers, the end-users of such models, these tasks can be challenging and time consuming.

This poster describes MLTUNE, a software tool-chain that automates much of the workflow for building ML models for performance tuning. MLTUNE is founded on the key observation that although many different ML-based models for performance tuning exist, at the core, these models share several common properties. For instance, feature vectors for most algorithms include either source code attributes [4], runtime performance events [2] or a combination [3]. By focusing on these common elements, we can abstract away the computational details in various steps in the workflow.

In MLTUNE we use the following formulation to describe a general ML model for performance tuning:

\[
M_p(\{F\}) \rightarrow \{T\}
\]

where, \(\{F\}\) is a feature vector containing static and dynamic program properties, \(\{T\}\) is a set of code transformation parameters and \(P\) is an objective metric such as execution time or energy. The task of \(M_p\) is to predict \(\{T_{opt}\}\) for a new feature vector \(\{F_{new}\}\) such that \(P\) is minimized (maximized).

Although fairly simple, this formulation captures the essence of almost all ML models for performance tuning that have been developed in recent years [1, 2, 3, 4, 5, 6, 7, 8]. Models for workload characterization can also be mapped to the above formulation by constructing \(\{T\}\) as a set of values describing a particular performance characteristic.

Based on the above formulation MLTUNE provides automated tools for feature extraction, feature selection, training data generation, labeling, validation and model selection.

2. WORKFLOW AUTOMATION

Fig. 1 gives an overview of MLTUNE. The framework is implemented in Python and leverages the scikit-learn package to implement the learning algorithms. In this section, we highlight the key automation steps.

2.1 Feature Extraction

Currently, MLTUNE can automatically extract any dynamic feature supported by the target platform. To capture these features, MLTUNE leverages the perf module which is standard on most Linux distributions. At install time, MLTUNE determines the number of measurable events that can be used as features. When a new model is to be built all measurable events are probed and these serve as the initial feature set. Measuring one event per program run can be time consuming given that there are hundreds of events and potentially millions of program runs. To address this issue, we include in MLTUNE a module that takes advantage of multiplexing to automatically determine subsets of performance events that can be measured during a single program run without causing conflicts in hardware counters.

2.2 Feature Selection

Selecting the right features is an extremely important step in machine learning. In most ML-based tuning work, features are typically selected by hand by performance ex-

---

1 We are unable to provide a more extensive list due to space constraints. We cite one work for each model category.
programs [5, 6]. Although effective in some situations, this ad-hoc approach can be limiting because not all attributes that influence the outcome vector may be known to experts. To avoid these situations, MLTUNE employs a series of automated feature selection techniques. In addition to normalization, standardization and scaling, the framework employs (i) subset selection with a greedy heuristic to gather features with highest ranking, (ii) clustering to eliminate strongly correlated and redundant features, and (iii) step-wise regression to filter out irrelevant features.

2.3 Labeling

Labeling can be a tedious and time consuming process. MLTUNE implements an algorithm that performs this task automatically. The roofline model [9] is used to establish upper and lower bounds for performance on the target architecture of a given code variant. The relative performance of each entry in the training data is then determined and ranked. A histogram is created based on the ranking and adjusted for the distribution of values. The buckets in the adjusted histogram form the classes for the target model and each entry in the training data is labeled accordingly.

2.4 Model Selection

Generally, it is not known a priori which model is most suitable for a particular instance. The choice of a model often depends on the characteristics of the training data. In MLTUNE, the generated training data is analyzed and a set of learning algorithms is selected based on the properties of the data. The selected models are passed through a suite of cross-validation tests and only the highest performing ones are presented to the user for testing.

3. STATUS

At the time of this submission, most component tools have been implemented and version 0 is available for download on github. We are currently working on the visualization module.

Initial experiments with MLTUNE have been quite encouraging. We have developed learning models for prefetching, thread migration and DVFS. Results from an automatically generated thread migration model is presented in Fig. 2.

4. REFERENCES


